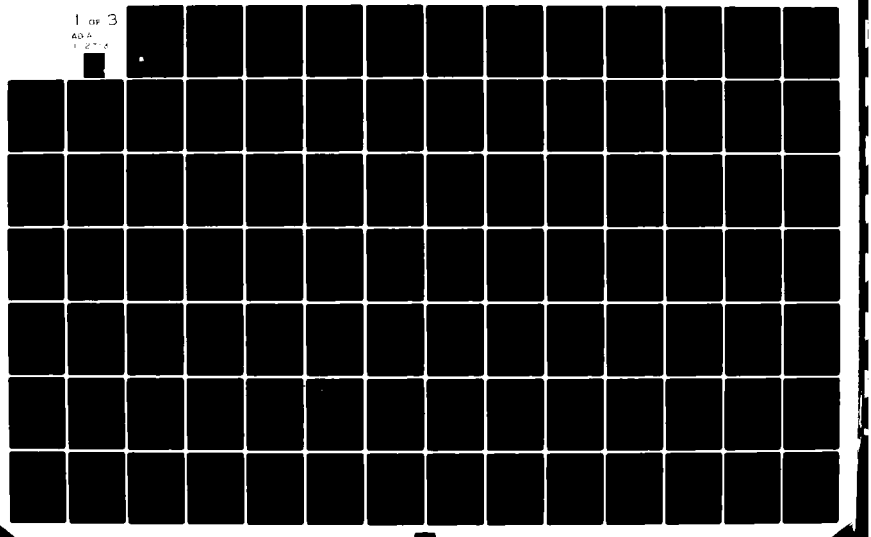
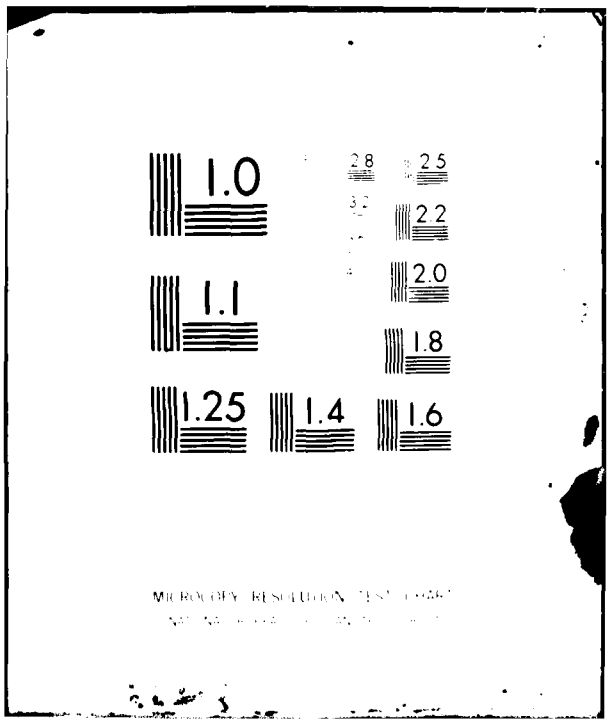
AD-A112 713 CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER --ETC F/G 9/2
AUTOMATIC GENERATION OF RELIABILITY FUNCTIONS FOR PROCESSOR-MEM--ETC(U)
FEB 81 V KINI N00014-77-C-0103
UNCLASSIFIED CMU-CS-81-121 NL

1 OF 3
AD A
112 713




1.0
1.1
1.25
1.4
1.6
2.0
1.8
2.2
2.5
2.8
3.2

CMU-CS-81-121

(12)

Automatic Generation of Reliability Functions for Processor-Memory-Switch Structures

Department of Electrical Engineering
Carnegie-Mellon University
Pittsburgh, PA. 15213
February 1981

Vittal Kini

DEPARTMENT
of
COMPUTER SCIENCE

Carnegie-Mellon University

CMU-CS-81-121

Automatic Generation of Reliability Functions for Processor-Memory-Switch Structures

Vittal Kini

Department of Electrical Engineering
Carnegie-Mellon University
Pittsburgh, PA. 15213
February 1981

Submitted to Carnegie-Mellon University
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

This work was supported in part by the National Science Foundation under Grant GJ 32758X and in part by the Office of Naval Research under Contract NR-048-645 and Contract N00014-77-C-0103.

ABSTRACT

Reliability computation is gaining much importance for computer system architectures with built in redundancy, such as multiprocessors. The task of computing the reliability function for arbitrary Processor-Memory-Switch (PMS) interconnection structures, however, is tedious and prone to human error. Existing reliability computation programs make one of two assumptions:

- That the case analysis of success states of the system has been carried out. Such analysis must be done manually. In this instance the input to the program is usually in the form of an intermediate representation (e.g. Fault Tree, Reliability Graph).
- That the interconnection structure is a member of, or can be partitioned into, some limited class of structures for which a parametric family of equations exists (e.g. N-Modular Redundant systems, Hybrid Redundant systems).

This thesis represents a first step in the development of a methodology for automating the computation of symbolic reliability functions for arbitrary interconnection structures at the PMS level. The work reported here automates the task of case analysis and problem partitioning in the hard-failure reliability computation for PMS structures. As a consequence attention is freed to focus almost wholly on specifying the reliability computation problem. The advantages of such an approach are (i) utility to a larger class of users, not necessarily expert in reliability analysis, and (ii) a lower probability of human error in the computation.

A program named ADVISER (Advanced Interactive Symbolic Evaluator of Reliability) was constructed as a research vehicle. ADVISER accepts as inputs

1. The interconnection graph of the PMS structure, and
2. A succinct statement of the operational requirements on the structure in the form of a regular expression.

Each component in the system, which may have internal redundancy, is represented by a symbol. The operational requirements in the case of a multiprocessor architecture may be, for example, "two processors and four memory boxes and one I/O channel". ADVISER considers the communication structures in the PMS system (e.g. buses, crosspoint switches, etc.) in addition to the explicitly stated requirements to determine how the interconnection structure affects the system reliability. The output of the program is a symbolic reliability equation for the system subject to the given requirements. This dissertation describes the ADVISER program and methodology in detail.

Acknowledgements

I would like to express my deep gratitude to Dan Siewiorek who has nominally been my advisor through my years of graduate study at CMU. As a good friend through hard times, and a patient and ever-helpful mentor he has gone far beyond the call of that nominal duty. Without his continual encouragement this thesis might truly never have been written.

I am much indebted to my present employers, USC-Information Sciences Institute, for being flexible and understanding in allowing me to pursue thesis writing in conjunction with day-to-day tasks over the better part of one and a half years. I am particularly grateful to Steve Crocker of USC-ISI in this respect.

For taking the time out from their busy schedules to review this large document in such a short time I extend my sincere thanks to my thesis committee: Mario Barbacci, Steve Director, both of CMU, and Bob Swarz of Digital Equipment Corp.

An undertaking such as this is seldom accomplished without the encouragment, support and contributions of good friends and colleagues; I am fortunate to have been blessed with many. I would especially like to acknowledge Mario Barbacci, Bill Brantley, Steve Crocker, Gary Leive, Andy Nagle, Alice Parker, and Don Thomas, all of whom, at one time or other, contributed ideas and acted as sounding boards during enlightening discussions. My thanks to John Gaschnig, now at SRI, for an introduction to the ideas which lead to the material of Chapter 4, and Rostam Joobbani for help in resurrecting old programs for use in the experiments of Chapter 7. The camaraderie of the "Porter Hole Gang" contributed to making the stimulating CMU environment that much more pleasant.

The Dunnoms, Jipps, Kerrs and Kondas provided homes away from home and affection beyond measure to a student far from his family and country. They have helped perhaps more than they realize. Last, but most of all, my eternal gratitude to my parents for their unwavering faith in my abilities and trust in my judgement. Would that all toilers were blessed with such a loving family.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

Recent years have seen the advent of practical multiprocessor architectures and distributed computer systems of growing sophistication. Their growth has been assisted by cheaper components, of far greater complexity and power than heretofore available, which have been produced by the revolution in techniques of large scale circuit integration. As the complexity and sophistication of computer systems grows, so does the importance of fault-tolerance and of system reliability as a design parameter. Formerly the province mainly of space-craft designers, fault-tolerance techniques are becoming commonplace due to the rising ratio of maintenance costs to initial capital costs in typical computer system life cycles. Thus computation of system reliability metrics have become part of the catalog of system design tasks. Various efforts have been reported in the literature and are in progress to provide designers with reliability design tools which will make the task of computing system reliability metrics easier and more efficient.

The computing of system reliability for complex multiprocessor architectures can be very tedious and quite prone to error, sometimes even for experienced reliability analysts. Software tools which currently exist to help estimate or calculate system reliability usually assume an understanding of reliability analysis techniques and are usually more in the nature of computational aids once the preliminary system decomposition and analysis has been manually achieved.

This dissertation describes the results of a feasibility study which was prompted by the question "Is it possible to build reliability design aids which will assume the burden of a significant portion of the system analysis effort leaving mainly the system reliability specification task to the designer?". The result of the effort was the ADVISER (Advanced Interactive Symbolic Evaluator of Reliability) program which accepts the interconnection structure of the architecture at the Processor-Memory-Switch level and a simple set of operational requirements on the architecture. It then produces the symbolic form of the system hard-failure reliability function under the given requirements. The program attempts to

analyze efficiently, using the divide-and-conquer paradigm, the various possible classes of cases of system success using information gleaned from the interconnection structure of the system. The program as it is currently constituted is not specific to computer systems *per se* and is applicable to other problems which can be cast into the same framework of assumptions. In other words, no semantics relating to the behavior of specific types of computer system components are currently incorporated into the program. The current scope and capabilities of the ADVISER program are modest but the methodology underlying its design shows much promise for building more sophisticated future versions.

Following sections will present a brief background on reliability calculation and a survey of some existing representative reliability calculation programs. The goals for ADVISER will be stated and compared with those of previous efforts. The final section will present the organization of this thesis.

1.1 Background

Computer systems may be studied at various levels of detail. Bell and Newell in their book on computer structures [Bell 71] proposed four broad levels at which attention is usually focused. These are respectively

- The gate level,
- the register-transfer level,
- the software level, and
- the Processor-Memory-Switch (PMS) level.

Reliability prediction studies at the gate and device level are concerned with large populations of identical components and their failure characteristics in field use. A useful compendium of such data on electronic components is to be found in [MIL-HDBK-217B 74]. A typical use of this data consists of deriving failure rates for components under various levels of environmental stress, component quality, etc. starting from a base failure rate in a benign environment. Some recent attempts in the development of reliability models for software are compared and contrasted in [Schick 78]. The problem of reliability prediction at the register-transfer level has traditionally been approached by considering individual gates, registers, flip-flops etc. to be subject to Poisson failures and computing the failure rate of the system by simply adding the failure rates of the constituent components. This sort of modeling is inadequate in the case of fault-tolerant systems since they contain redundancy.

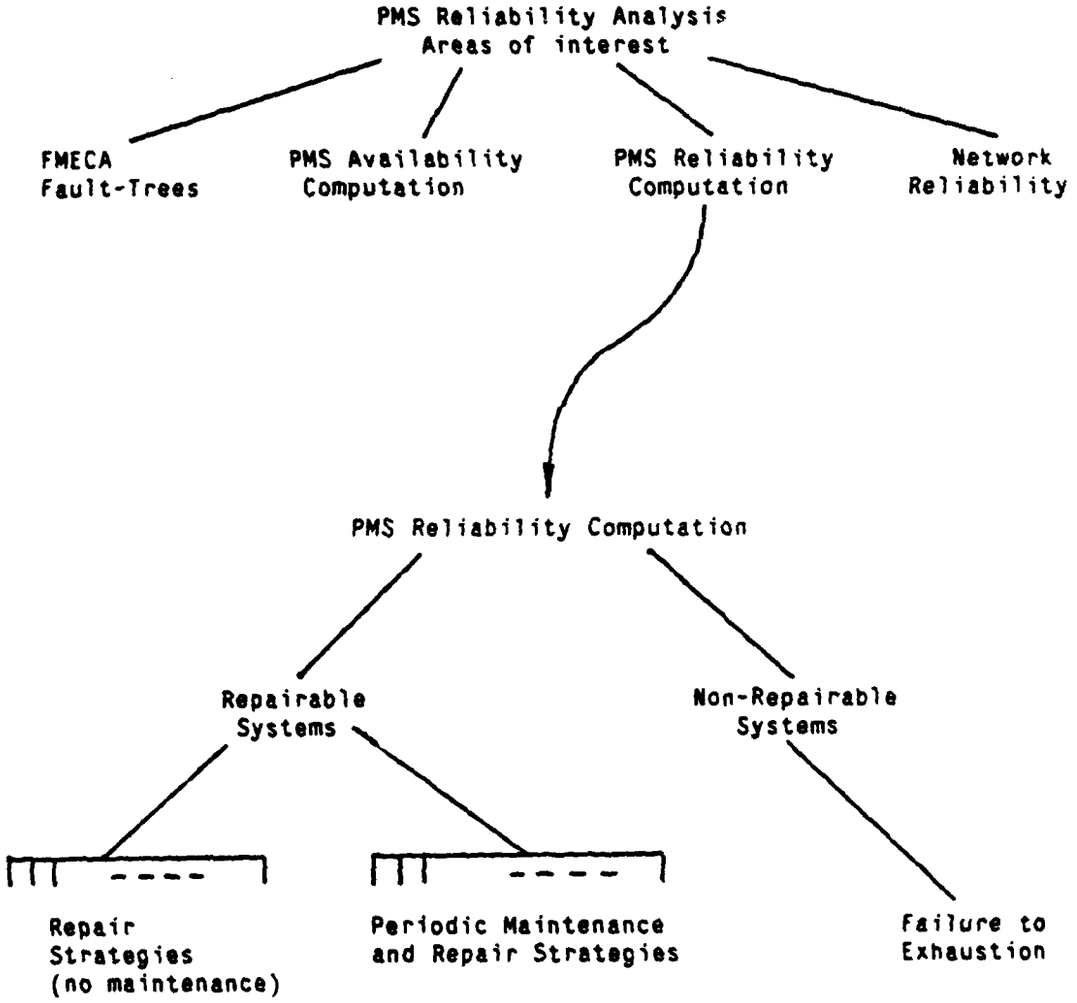


Figure 1-1: Reliability modeling at the PMS level.

We shall be concerned here with the Processor-Memory-Switch or "system" level of detail of computer systems. Figure 1-1 shows the areas of reliability assessment which are of broad interest at the PMS level. These categories are not necessarily disjoint and serve only to grossly characterize the distribution of work reported in the literature.

FMECA or Failure Modes, Effects and Criticality analysis ([Greene 68]) attempts to enumerate and explicate all the failure modes of a complex system and seeks to understand all the origins and manners of progression of various sequences of primal failures which could lead eventually to system failure. This form of analysis is particularly useful in studying systems whose failure can have disastrous consequences since it forces designers of complex systems to consider unusual failure sequences which might otherwise have been overlooked. Furthermore, once completed, the analysis serves as a form of detailed system documentation. Fault Trees are a form of data representation which is invaluable in this class of analyses. Though not dealing with computing systems, a well known example of the use of FMECA and fault trees is the Rasmussen Report [USNRC 75]. Lapp and Powers [Lapp 77] describe a methodology for automatic synthesis of fault trees for chemical engineering systems (see below).

Reliability analysis is concerned with obtaining the system reliability function [Shooman 68]. The reliability function for a system is a function of time and gives the probability that the system will have survived upto a given time without failure since time zero. If the system reliability function is known the mean time before failure (MTBF) may be calculated by integration. For non-repairable systems such as spacecraft the time to first failure is effectively the lifetime of the system. The knowledge of the reliability of two non-repairable systems allows the computation of the Mission Time Improvement (MTI) factor [Avizienis 75] which serves as a comparative measure of system usefulness. The MTI is the ratio of the mission times at which the two system reliabilities decay to some pre-specified minimally tolerable value. Also of interest is the reliability importance [Barlow 75a] of a given system component which is roughly the sensitivity of the system reliability to the component's reliability.

Availability analysis is concerned with obtaining the system availability function [Shooman 68]. The availability function of a system is a function of time and gives the probability that the system will be operational at a given time. The system may have been subject to earlier failures and subsequent restoration to operation after repair. The introduction of a second random variable into the picture (i.e. the repair time) complicates the mathematical analyses and some sort of Markov analysis becomes essential in order to obtain the availability function

of the system. Even the Markov models are not amenable to extraction of a closed-form solution except in a limited number of cases. Usually, however, the limiting system availability is sought since it is possible to calculate it combinatorially from the limiting availabilities of the individual components. The limiting availability serves as a measure of the fraction of time the system will be available for use over its lifetime. For non-repairable systems the availability is synonymous with reliability. Availability analyses are important in the computation of system life-cycle costs.

The so-called "network reliability problem"[1] is concerned with calculating fairly simple measures of reliability for a system. Typically the system is a computer communication network and the vertices of the interconnection graph denote the computers while the arcs denote the communication links. Either arcs or vertices, or both, are assumed to fail stochastically. Typically, all vertices are considered homogeneous with identical probabilities of failure. Arcs are also typically treated likewise. Two common reliability measures computed for such a system are, for instance,

- The probability that some specific pair of vertices will have at least one communication path between them at all times.
- The probability that the operative arcs always contain a spanning tree of the network.

[Wilkov 72] is a good tutorial paper on the subject. Despite their seeming simplicity these types of network reliability calculation problems have been shown to be NP-hard in the case of general networks (see [Rosenthal 77] and [Ball 80]).

1.2 Extant Reliability Calculation Programs

In this section we shall review a few representative extant computer programs for system reliability computation. Their characteristics and intent will be briefly discussed with a view to setting up a framework within which to classify ADVISER.

---

[1] The term "network reliability problem" is applied in the literature to two distinct kinds of network problems. The first variety deals with graph models of computer communication networks as in [Wilkov 72] and [Hansler 74]. The second variety addresses two-terminal directed networks which are essentially reliability graphs. Such a graph is *not* necessarily a model of the physical interconnection structure of the system but rather is a *representation* of it which characterizes the system's reliability. We shall use the term "network reliability problem" exclusively in the sense of [Wilkov 72].

1.2.1 Reliability Estimation

Nelson, Batts and Beadles [Nelson 70] describe a program which computes the bounds on system reliability given its reliability graph. An upper bound for the system reliability is given as the sum of the probabilities of functioning of the path or tie sets of the reliability graph. A lower bound for the system reliability is obtained by taking the first two terms of the finite series which gives the probability of the union of several events, namely:

$$Pr\{E_1 \cup E_2 \cup \ldots \cup E_n\} = \Sigma_i Pr\{E_i\} - \Sigma_{K_j} Pr\{E_i \cap E_j\} + \Sigma_{K_j K_k} Pr\{E_i \cap E_j \cap E_k\}$$

$$\ldots\ldots + (-1)^{n-1} Pr\{E_1 \cap E_2 \cap \ldots \cap E_n\}$$

where the event E_i is in this case the functioning of all components in the i^{th} tie set. Increasingly tighter upper and lower bounds on the system reliability can be obtained by taking more terms of the expression above. Similar bounds can be obtained on the unreliability by considering E_i to be the event that all components fail in the i^{th} minimal cut set of the reliability graph. The existence of the system reliability graph is assumed. Components are assumed to have constant reliabilities. Matrix methods are used to generate the minimal cut sets of the graph. Bounds based on the tie sets are recommended in the low reliability region and those based on cut sets are recommended in the high reliability region.

1.2.2 Reliability Block Diagram representation

The exact combinatorial system reliability derived from the reliability block diagram representation is the subject of [Fleming 71], [Chelson 71] and [Kim 72]. All these efforts assume that the system reliability graph (in the form of a reliability block diagram) has been previously derived by the analyst. Fleming [Fleming 71] describes a program named RELCOMP which computes the system reliability and MTBF. The program accepts what is essentially a purely series reliability block diagram. RELCOMP assumes that the system is composed of independent subsystems which fall into one of eight categories provided for e.g. standby redundant configuration, actively redundant configuration, etc. The corresponding eight commonly used reliability equations are built into the program. Both exponential and weibull failure distributions are represented in the equation repository.

Chelson [Chelson 71] describes a program which accepts a particular form of block diagram able to represent systems with standby redundancy. More than one block may represent a given system component and these are called equivalent blocks. Exponential

failure distributions are assumed throughout and different failure rates may be assigned to spares and active modules. The switches which represent the recovery capability of the system may be modeled as being imperfect. The program constructs the probability tree (Chapter 3) for the system and computes the reliability from it.

Kim et al. [Kim 72] describe a method for computing reliability from non series-parallel reliability block diagrams. Their procedure consists of three steps: (i) Reduction of all series and parallel connections until the block diagram cannot be reduced further ([Krishnamurthy 72] describes another reduction method), (ii) Enumeration of all paths from source vertex to sink vertex in the block diagram, and (iii) Computation of system reliabilities from the path reliabilities using an operation which amounts to counting the probability of a given component only once in each product term. Matrix methods are used to compute the paths sets for the block diagram.

More recently, work has been reported on the use of reliability graphs to produce symbolic system reliability functions ([Satyanarayana 78], [Aggarwal 78]). These results could be applicable with modifications in the case of ADVISER as described in Chapter 7.

1.2.3 Hybrid-Redundant System analysis

Another class of programs for system reliability analysis focus on weak points of the purely combinatorial analysis technique i.e. the inability to deal with systems containing varieties of dynamic redundancy [Avizienis 75]. In such systems the switching in of spares to replace failed modules is viewed as an imperfect process contrary to the assumptions of static reliability models. In such "staged" systems the so called coverage factor, or the probability of system recovery after a fault, is of central importance since the system reliability has been shown to be very sensitive to the factor [Bouricius 69].

The early effort in this instance was the REL program which was succeeded by REL70 [Bouricius 71] written in APL. Bouricius, et al. derived basic equations for systems with standby sparing largely under the assumption of constant failure rates for all system components. The coverage factor, C, was included in these equations and it was shown that assuming perfect coverage (C = 1) even when coverage was in fact "near" perfect (C = 0.99) could produce gross errors. The results of this work were incorporated as an equation repository into REL70 to analyze memory and processor subsystems of a typical computer.

Mathur [Mathur 72] describes a computer program named CARE which was an

improvement on REL70. Systems being analyzed were viewed as cascades of independent hybrid-redundant subsystems. Again, a repository of equations was built into the program for the analysis of each of such subsystems. Equations developed in [Bouricius 71] were also included. The system reliability was taken to be the product of the independent subsystem reliabilities. The latest version of the program, CARE III, developed by Raytheon Corp., is considerably more complex. A Markov process approach has been incorporated into the program along with decomposition methods which agglutinate states to reduce the large state space of a complex model. Time-dependent parameters for transitions between the states of the Markov model (i.e. a non-homogeneous Markov model) are handled in cases of non-repairable systems. Since ultrareliable systems are the subject of CARE III much attention has been paid to reducing numerical error.

More recently, Ng and Avizienis ([Ng 77], [Ng 80]) developed a unified reliability model for fault-tolerant systems. This model is based on a Markov process view of the graceful degradation process of dynamically redundant systems. Various earlier reliability equations derived for different types of static and dynamic redundant systems are available as special cases of the unified model [Ng 80]. In addition the model is extended to derive the reliability of repairable systems under a restricted model of the repair process. Degradation under transient faults is also modeled by the same Markov techniques. The ARIES program embodies the results of the unified model. However, the model is still restricted in its applicability to those types of systems which are decomposable into cascades of independent hybrid-redundant subsystems.

Landrault and Laprie [Landrault 78] describe the SURF program which views repairable systems as being governed by non-exponential failure processes. The Coxian device of stages [Cox 68] is used to judiciously introduce series of fictitious states with exponentially distributed transition times among them so as to convert the non-Markov process to a Markovian one in the cases where non-exponential distribution being considered is related to the exponential (e.g. Gamma, Erlang etc.). For some problems semi-Markov processes are also used which suppose the existence of a finite number of instants possessing the property of independence on past history i.e. an imbedded Markov chain.

1.2.4 PMSL

A quite different view of Processor-Memory-Switch (PMS) systems is contained in [Knudsen 73]. Knudsen describes PMSL, a language and a system to describe arbitrary PMS structures. The notation developed in [Knudsen 73] is quite similar to its progenitor the PMS notation of Bell and Newell [Bell 71]. PMSL was programmed in SNOBOL and was a powerful description facility which allowed users to construct interconnection models of arbitrary PMS structures with the program doing various attribute checks on the structure for legality of interconnections. The PMSL system, although more in the nature of a PMS-database manipulation system, also allowed the user to compute the combinatorial reliability of the PMS structure input to it. However the program suffered from very rudimentary reliability calculation facilities. Reliability calculation was applicable only to uniprocessor structures and enumeration of system success states was used as the (inefficient) computation method. PMSL is included in this survey of reliability computation programs because the level of detail in its model of PMS structures is similar to that in ADVISER although the instruments provided to manipulate PMS descriptions are more powerful in PMSL. PMS structures are viewed as being hierarchical and components in them are each described by a list of attribute-value pairs.

1.2.5 Automatic Fault Tree Synthesis

We end this brief survey of PMS reliability computation programs with a look at an example from the field of Chemical Engineering. Although not entirely relevant to computer systems, this example is important since it is a step toward the eminently desirable goal of easier and less error-prone reliability computation for complex systems. Lapp and Powers ([Lapp 77], [Powers 76]) describe the FTS program which constructs the fault-tree representation of a complex chemical engineering process from a much simpler logical model of the process. The program contains hazard models of commonly used pieces of equipment within the process (e.g. valves, pumps, sensors, reactors etc.). The user constructs a logical flow diagram of the process, labeled with various process parameters, and the program uses its database of hazard models and information to synthesize the fault tree for the process. Using cut set analysis the probability of the top event or system failure may be computed.

1.3 Statement of Goals and Discussion

In the construction of ADVISER the goal was to produce a reliability calculation program capable of computing the symbolic reliability function for an arbitrary PMS interconnection structure given a simple statement of the operational requirements placed on it. Therefore, the following ends were pursued.

- The program should require only a modicum of information from the user as input i.e. the specification of the problem should be simplified.

- The program should attempt to assume the major portion of the analysis of the interconnection structure preparatory to computing the reliability function. This will make it attractive to the user who is less experienced in reliability analysis and the chances of human error creeping into the computation will be reduced.

- The program output should be the symbolic system reliability function so that arbitrary failure distributions for the individual component reliabilities may be experimented with.

One of the major emphases in ADVISER was to avoid the manual construction of the reliability graph or equivalent representation of the system thus making it preferable to programs such as described in [Chelson 71] and [Kim 72]. Also emphasized is the observation that since knowledge of the physical interconnection structure provides information about the structural dependence (as distinct from statistical dependence) of components in determining the system reliability, the operational requirements on the structure can be expressed very simply in terms of a few key components in the system. Further information can then be deduced from the interconnection topology. This leads to the succinct statement of minimal system requirements in the ADVISER paradigm.

Of interest in the investigations were systems which could not be partitioned into independent hybrid-redundant subsystems as assumed in [Mathur 72] and [Ng 80]. Examples of such systems are the Pluribus [Ornstein 75], Cm* [Swan 77], and Tandem-16 [Katzman 77] multiprocessors in which recovery from faults and reconfiguration is done largely by software or firmware. This is not to exclude the possibility that, say, one of the processors within a multiprocessor such as Tandem-16 could be constructed for reliable operation by using hybrid redundancy internally. The difference is one of the level of detail at which the system is being studied.

Network reliability analysis of the form addressed in [Hansler 74] and discussed above in Section 1.1 was only of marginal interest. The reason is that PMS structures are more closely-coupled than computer communication networks and the operational requirements on them

are usually more complex than in the kinds of problems studied in [Hansler 74]; Chapter 7 shows that ADVISER can be used for a subclass of the latter kind of problem.

The ADVISER program was aimed more toward solving the common problem of deriving the combinatorial reliability of complex interconnection structures under various operational requirements, particularly in the context of comparative reliability studies of PMS interconnection structures. A possible use of ADVISER is in an iterative design study of a candidate PMS interconnection structure wherein the structure topology is perturbed, components added or deleted, etc. until the appropriate reliability is achieved.

1.4 Organization of Thesis

This thesis is divided into eight chapters and two appendices. Chapter 2 presents an overview of the ADVISER program and introduces some underlying concepts, definitions and terminology which are used in the remaining chapters of the thesis. In effect Chapter 2 is a version of the thesis in miniature. It is provided so that the reader may have a backdrop against which to understand the detail in subsequent chapters and as such it is recommended reading. Chapter 3 describes the intermediate representation used by the program to maintain the results of its intermediate computations. This is logically equivalent to the block diagram representation described above, the difference is that it is not manually constructed. Chapter 4 discusses algorithms on the PMS interconnection structures for detecting symmetries which could be of use in reducing extraneous computation. Chapter 5 describes a class of subgraphs of PMS structures and the special reliability computation techniques which were derived for them. Chapter 6 presents details about the Overlord routine within ADVISER. This embodies the reliability evaluation paradigm and controls the rest of the program parts. Chapter 7 describes experiments carried out with ADVISER in order to test the models the program generated. Chapter 8 summarizes this work and presents directions for future efforts. Appendix A describes a special case encountered by the algorithms of Chapter 5. Finally, Appendix B presents a list of terminology and acronyms used in the dissertation.

Chapter 2
Overview of ADVISER

2.1 Underlying assumptions and concepts

The general case of deriving reliability functions for arbitrary interconnection structures of components is a task that is difficult to program. Much depends on the semantics of the behavior of the components in the structure, the interrelations among their individual tasks within it, whether their probabilities of functioning are statistically *mutually independent*, and so on. Some idealizations become necessary in order to make the problem tractable.

One of the original and more important goals of the project was to produce a hardware design tool. The desire was to be able to compare two PMS *interconnection-structure* designs with fast turn-around time. As long as the metric used is consistent across the space of designs being considered, the comparison is valid. For this reason it was decided to study the hard-failure reliability of a system unencumbered by the effects of policy decisions regarding manner of use, software reliability, transient failures, and statistical dependence of component *failures in any form*. The comparisons would therefore take into account the best possible reliability performance of each PMS structure being considered.

In order to set a reasonable goal for this thesis certain fundamental assumptions were made and limitations set.

1. To begin with, failure processes in individual components in the structure were assumed to be stochastically independent. Since Processor-Memory-Switch structures are the focus of the study, there seems to be justification in making this assumption. For example, the typical components we are considering, such as processors, memories etc., are generally physically separated. Thus common-mode failures caused by proximity, such as heat generated by one component causing thermal runaway within another, would have lower likelihoods. Dependency of failure mechanisms was considered a second-order effect. This, however, does not imply that failures of different system components affect the system uniformly. Clearly, the topology of the interconnection in the structure has a bearing on this question.

2. Only the hard-failure reliability function of the components is addressed. The effect of variation in coverage [Bouricius 69] are not considered and neither are transient failure mechanisms.

3. Components in the PMS structure will be assumed to be binary state objects, i.e. either "failed" or "working". This assumption, by implication, once more excludes consideration of transient failures. Furthermore, the emphasis will be on probabilities of success of components so that all reliability functions will be expressed in these terms.

4. The graph of the interconnections of the PMS structure will be modeled as a non-directed graph. The vertices of the graph will correspond to the components in the structure and the functionality of the components will be lumped into these vertices. Each non-directed arc of the graph will be considered perfectly reliability and will simply represent the capability of information to flow between its two end vertices. The failure of a component is assumed to preclude its being able to process, and, more important, retransmit any information sent towards it, when it is in its failed state. This is equivalent to removing the corresponding vertex in the graph and all arcs that are incident on it.

5. *It is assumed that in order for an assemblage of information-processing components to comprise a useful functioning system, some distinguished set of critically important system components will need to be able to communicate amongst themselves. In other words, information should be capable of flowing between any two components from the distinguished set; whether via other distinguished components or any other components in the structure, or both.* This will be henceforth referred to as the Communication Axiom and is elaborated upon below.

We first introduce some concepts basic to our discussion of the Communication Axiom. Throughout the rest of this dissertation the terms "system success" and "component success" will be used interchangeably with the terms "system is functional" and "component is functional" respectively, i.e. to denote the state of not being failed. In any system that we may consider, some subset of the total set of components in the system will be distinguished in that their functioning correctly is of vital importance to system success, e.g. the CPU in a uni-processor system. More accurately, there will be a set of generic types of components of vital importance (e.g. *processors and memories*). Also, a certain minimum number of components, drawn from each distinguished type will be required to be functional for system success. The distinguished component types will be termed critical component types (CCTs) Each such type constitutes a class of *identical* components and the members of these classes will be termed critical components. All components that are not critical in the PMS structure will be termed auxiliary components. For instance, consider the simple example of Figure 2-1 where we depict two processors which pass data back and forth over one of two links provided to increase reliability of data transfer. We shall assume that the data links are very

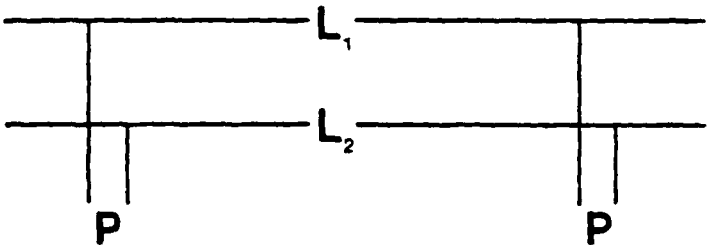


Figure 2-1: Critical and Auxiliary components

different in their reliability behavior thereby essentially being members of different components type classes. Assume the processors are vital to the task at hand and are thus critical components, then links L_1 and L_2 are auxiliary components in that there are system success states in which L_1 is functional but not L_2, and *vice versa*, and a system success state in which both L_1 and L_2 are functional. However, there are no system success states in which critical components are not functional.

A minimum number of critical components from each CCT are required for system success. Together they constitute a minimal critical resource set (henceforth MCRS). The set is minimal in the sense that, although the system may function if all components in an MCRS are functional (depending on the status of the auxiliary components in the structure), the structure is guaranteed to fail if any component of this MCRS fails. In other words, the success of an MCRS is a necessary, though not sufficient, condition for system success. This concept is not to be confused with a *minimal* system success state in which the failure of any one functioning component, whether critical or auxiliary, causes system failure. The latter would be a stronger condition on minimality.

If there is redundancy in the supply of critical components configured in the structure then there will be more than one minimal critical resource set. Each such set will, in general, be included in one or more system success states, again depending on the disposition of the auxiliary components.

We now continue our discussion on the Communication Axiom. It seems fundamental that, in order for an information processing system to do useful work, there need to exist pathways, or channels, of information flow between components of an MCRS of that system. This is a basic rule which is tacitly assumed during calculation of reliability of Processor-Memory-Switch structures.

Notation: In the rest of this dissertation the interconnection graph of the PMS structure under study will be referred to as G(V,E).

The contention here is that the reliability of PMS structures may be computed by a program using the following simple paradigm. The user inputs:

1. Component type classifications
2. Graph of the PMS interconnections, and
3. A Boolean statement of which component types are critical component types and how these are related in determining system reliability.

The last of these three items is what we shall term the minimal requirements on the system. An example of such a minimal requirement phrased in English might be "at least one processor *and* at least one memory *and* (at least one disk *or* at least two tape units) must be functional" where the "or" is an Inclusive-OR. Of course, the program would use some abbreviated or encoded form of such a statement. The program would employ the minimal requirement and the interconnection graph of the system in the context of the Communication Axiom. The component types referred to in the minimal requirements would be labelled critical component types by default and the rest labelled as auxiliary types. The minimal requirements would be used to generate all the MCRSs of the system. For a given MCRS, the Communication Axiom and the interconnection graph then identify sets of paths between pairs of vertices in the graph which represent the components of the MCRS. A path is deemed functional iff all the components along that path are functional. The Communication Axiom implies that components in the MCRS must be part of a connected graph of functional paths for a reliable system. A more precise statement of the Communication Axiom is given in Section 6.6.1.

In order to gain an intuitive understanding of how the Communication Axiom is used each MCRS may in essence be thought of as a skeleton of critical components which must be "fleshed out" with a set of paths in the graph between the vertices of the skeleton so as to form a connected graph. This will provide paths for communication between the components in the MCRS. Each such possible fleshing out of the skeleton will correspond to one minimal success state of the system. *Furthermore, each set of paths chosen to flesh out the skeleton will identify the other (auxiliary) components along those paths which are additionally necessary for the vertices of the MCRS to communicate.* This method of identifying the additionally necessary components has an important and useful side effect from the viewpoint of the user of the program. Consider a component in the structure whose component type is not referred to in the requirements expression. It may be the case that this component is required to be functional in every system success state, i.e. it is truly a "critical" component

although it has been labelled auxiliary by default. An example of such a component might be a memory controller which lies on the path to a memory required for system success by a requirement of the type shown above. Although the memory controller is not referred to in the requirements, thereby not explicitly making it critical, the strategy of "fleshing out" the MCRS with paths from the graph will always find the memory controller to be necessary at all times since all paths to the memory pass through it. Typically, therefore, few component types will need to be explicitly labelled critical by including them in the requirements expression since other critical components will be deduced from the interconnection structure via the path-finding strategy.

On the basis of the foregoing discussion it is possible to see that each MCRS will be part of possibly several system success states depending on how many combinations of paths can be found which flesh out the skeleton it provides. For each MCRS a reliability expression would be generated which accounts for all the probabilities of the all the functional states of which the MCRS could be a part. These reliability expressions will henceforth be referred to as Intermediate Results or Partial Results (see Chapter 6 for a more precise description of these). The intermediate results relating to all possible MCRSs would finally be combined in the appropriate fashion to generate the system reliability expression.

The use of the Communication Axiom, in the manner referred to in the paradigm outlined above, seems to be sufficient to derive the reliability function for many cases of arbitrary PMS interconnection structures. However, constraints beyond those implied by the Communication Axiom are sometimes posed, during calculation of PMS system reliability, by the special types of behavior exhibited by various system components. For example, a crosspoint switch, unlike a bus, generally allows communication only between components connected to distinct sides of the switch and not among the components connected to the same side. It would be impractical to include in the program all the semantics of various types of special behavior ever to be encountered, although this might be reasonable for a limited set of special component types. However, it was postulated that the inclusion of three further types of simple modeling information as additional side-constraints as inputs would enable the program to handle a majority of cases. This keeps the model and the required operations simple while providing a useful tool in the program. The three types of additional information are named below and will be explicated in detail in Chapter 6. However, a preliminary discussion of them will ensue in Section 2.2.1.3

- The internal port-connection matrix of a component.

- The possibility of intra component-type information transfer.

- The required clustering of functioning critical components in parts of the PMS structure.

2.2 Overview of program

This section will present a fairly lengthy outline of the process by which a PMS description and the associated reliability requirements upon it are operated on by the ADVISER program to produce a symbolic reliability function. The process, of course, is subject to the assumptions and limitations set forth in Section 2.1. This overview is intended to provide a broad picture of the program within which its individual parts may be described in detail without much repetition of information to set the context for the description. The material in subsequent chapters of this thesis will elaborate on the various stages of the program and lay out implementation issues and details. First, however, some insight into the structure of the program and the nature of its input is desirable.

When calculating a reliability measure for a system of components, three items of information are necessary, namely:

1. The reliabilities of the individual components in the system,
2. The physical or logical connection of those components which give the system its particular existence and define its reliability, and
3. The operational requirements placed on that system which affect its perceived reliability; for, clearly, a multiprocessor, say, would be less reliable in the case of a task which requires any m of its processors to be functional as opposed to the case where a task requires only any $n<m$.

Figure 2-2 illustrates the structure of the ADVISER program and its various phases. This figure is reproduced in Figures 3-1, 4-1, 5-2 and 6-1 with enhancements to indicate which part of the program structure is addressed by the corresponding Chapter. Subsequent sections in this chapter will describe the above three kinds of input into the program and their eventual use.

2.2.1 Program Inputs

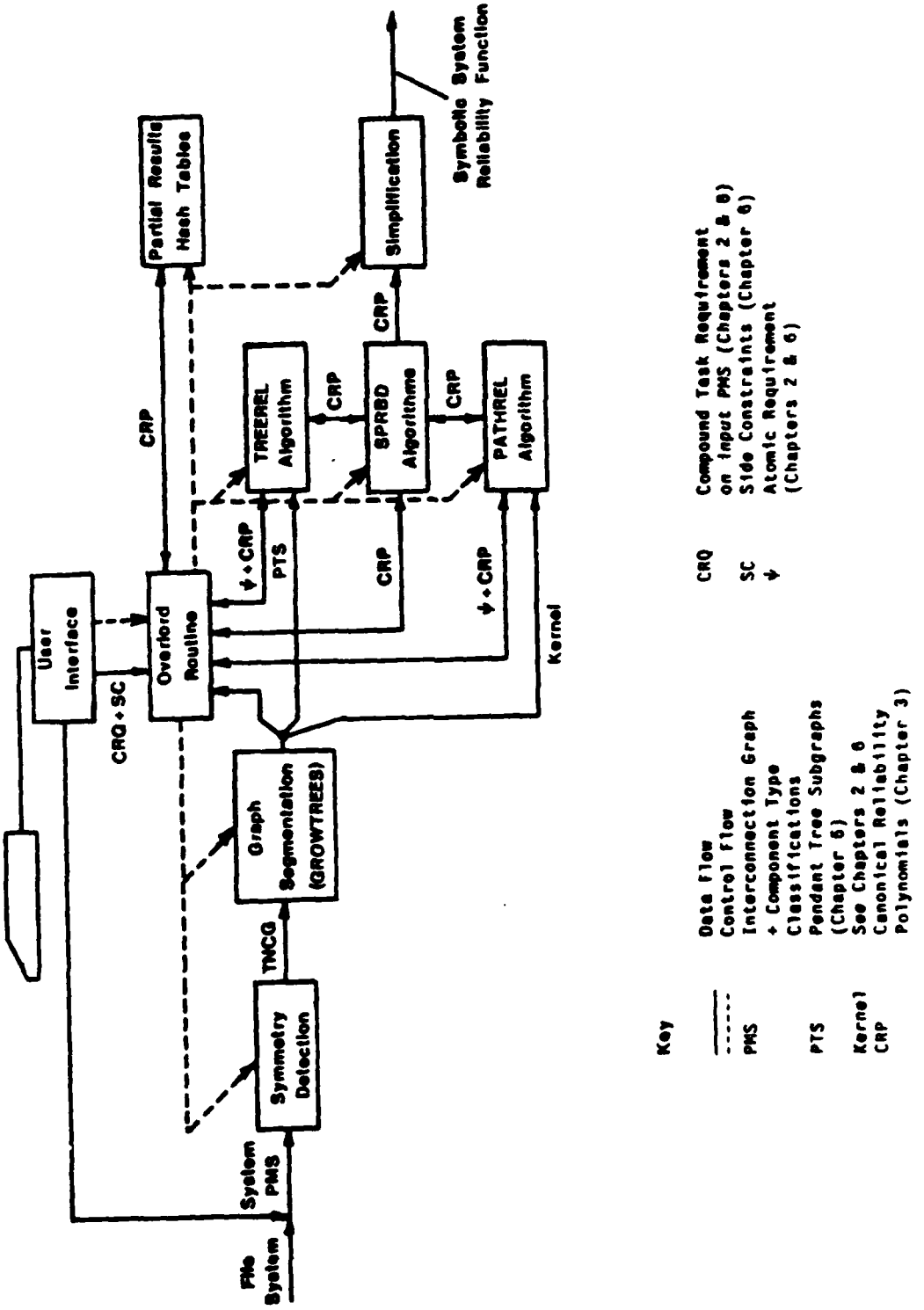


Figure 2-2: The structure of the ADVISER program.

2.2.1.1 Declaration of Component Types

The first input to the program is a list of types of components that will comprise the PMS structure yet to be described and for which the reliability function is to be computed. Each type-declaration will contain information with respect to the reliability function for a component of that type; whether it be a function type known to the program or whether it is some user-defined function elsewhere. The type-declaration will also contain a "print-name" which is to be used to represent the component when the reliability function is printed out. The reader will have gathered by now that when the interconnection structure is defined the components comprising it will each be assigned a type which may be selected only from this list of type declarations. The outcome is that components of like type are assumed in the current implementation to be identical in a reliability sense (see below for a discussion of another option). In other words, they are drawn from the same population. As an example, consider the following representative type declarations shown in tabular form (Table 2-1).

type #	typename	printname	rel.fn.	rel. fn. parameters
...	...	...	...	...
...	...	...	...	...
3	Cent.Proc	PC	Exponential	Lambda = 200.1/MHr
4	IO.Proc	PIO	Weibull	Scale = 385.3/MHr, Shape = 0.86
...	...	...	...	...
6	IO.Cont	KIO	Weibull	Scale = 286.7/MHr, Shape = 0.92

Table 2-1: Sample input component type-declarations.

The unique type numbers in the first column are assigned by the program. Components which are labelled as belonging to type 3 are of type Cent.Proc (central processors) whose factors, in the reliability function produced eventually, will printout as PC. All "Cent.Proc"s are declared to be identical and to have exponential reliability functions with a failure rate of 200.1/MHr. All type 4 components, likewise, are of the class IO.Proc (input/output processors) and have Weibull reliability functions, each with a scale factor of 385.3/MHr and a shape factor of 0.86.

Assigning a type to each component in a PMS structure may be viewed as imparting a label to the vertex representing that component in the interconnection graph. This information is used by the program as a constraint in detecting structurally symmetric subgraphs of the interconnection graph (see Chapter 4). The motivation for detecting such symmetries is, of course, the expectation that the amount of necessary computation can be reduced (see Chapter 6).

As stated above, the current implementation of ADVISER views all components classified as belonging to a given component type as having identical reliability functions. Another option is to relax this restriction and use the type-classification mechanism solely to classify the system components to assist in symmetry detection. The reliability function of each component would then be individually referred to, as and when it became necessary, rather than inferring it from its component type class. Doing this would allow further flexibility in the use of the system reliability function when it is generated (see Section 2.2.4). Trivial changes are required in ADVISER software to effect this.

2.2.1.2 Declaration of the PMS structure

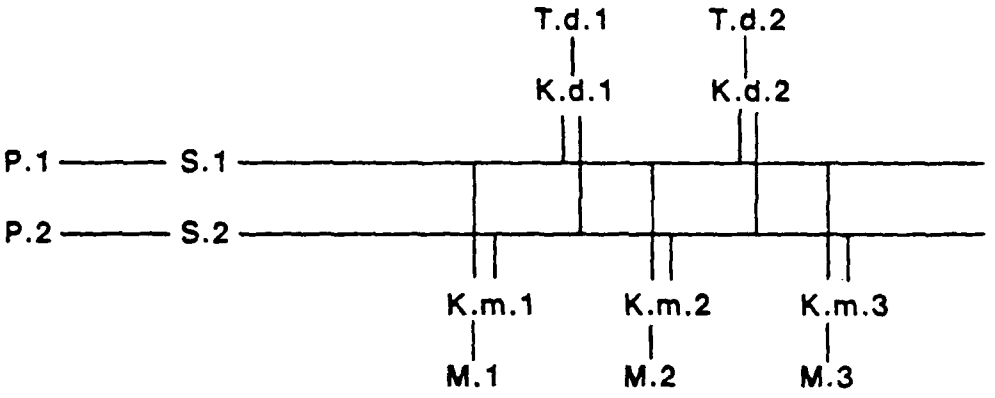
The next type of input to the program is the labelled graph which represents the interconnection topology of the system components. The model of interconnections underlying this work was described in Section 2.1 as representing all connections between components as duplex, i.e. information may flow in both directions along a connection or arc in the graph. Thus the model uses non-directed graphs. The description of the graph in the program input is achieved very simply by means of an adjacency list. A section of a typical graph description input is shown in tabular form below (Table 2-2). Again a description of the actual syntax is deferred for reasons of clarity.

component #	component name	component type	neighbor components
...	...	...	...
...	...	...	...
3	P.IO.1	IO.Proc	Unibus.1,K.IO.1,K.IO.2,...
4	K.IO.1	IO.Cont	P.IO.1,DISK.1,DISK.2,...
5	K.IO.2	IO.Cont	P.IO.1,TAPE.1,TAPE.2,...
...	...	...	...
...	...	...	...

Table 2-2: Sample inputs defining PMS interconnections.

In the table the component named P.IO.1 is declared to be of type IO.Proc. This component type must have been declared during the first input phase when component types were specified. K.IO.1 and K.IO.2 are seen to be declared identical components both with reliability functions which are Weibull with scale factor of 286.7/Mhr and shape factor of 0.92 (refer to Table 2-1). It is possible to completely specify an arc in a non-directed graph by one occurrence of one of the arc's two end vertices on the adjacency list of the other end vertex. However, it will be noted that in the program input each arc must occur on two adjacency lists. Thus, for instance, though it is enough for K.IO.1, in the example above, to appear on

the adjacency list of P.IO.1 to deduce that an arc exists between them, P.IO.1 must also appear on K.IO.1's adjacency list. The reason for this redundancy is twofold. Firstly, it ensures that the underlying graph model is adhered to. It enables the program to discover instances of errors in connections which manifest themselves as one-way links between a pair of components. Also discovered, are errors wherein the name of a component, not explicitly declared, appears on some adjacency list, thus representing a connection to a non-existent component. Secondly, a reader of an input file is more easily able to understand the structure of the interconnection graph if the connection is made quite explicit with two-way links. However, this restriction is easy to remove if the underlying model were to be changed to use directed graphs.



Key

P	Processor	K.m	Memory Controller
S	Processor Bus	T.d	Disk Drive
M	Memory	K.d	Disk Controller

Figure 2-3: Example PMS structure for explanation of requirements input.

2.2.1.3 Declaration of Reliability Requirements

We now come to the third kind of information necessary to calculate system reliability; a statement of what subsets of what types of system components need to be functional before the system is considered functional. In other words, what subset of system resources are required to be functional before the given task runs to completion on the system. This

information can be supplied in a variety of ways and an example will help to make the subsequent discussion clearer.

Figure 2-3 shows a dual-processor system. Each processor accesses memory and peripherals over a bus (S). The perhipherals are dual-ported for access from both processors. Let us assume, for a specific task, that at least one of the processors, at least two of the memories and at least one disk drive, need to be functional for system success.

One way to convey these requirements is to explicitly enumerate the system states which are success states. For instance, in our example, {P.1, S.1, K.m.1, M.1, K.m.2, M.2, K.d.1, T.d.1} is a full specification of one system success state. The program then has only to sum up the probabilities of occurrence of each state. This is objectionable for two reasons. Firstly, the number of system success states can be large for systems of reasonable size. However, an argument can be made that only that subset of the system success states which consists of minimal success states need be considered.

> **Definition 2.1:** A system is defined to be in a minimal success state when it is functional even though some components are failed, however, the subsequent failure of any one functioning component causes the system to fail, cf. the minimal cut vector in the terminology of [Barlow 75a].

The probabilities of all states which are subsumed by the minimal success states will cancel in the summation process. Even so, secondly, asking the user of the program to analyze and supply the set of minimal success states is objectionable. It is tantamount to asking him to do a major part of what is viewed as a task which ought to be done by the program to justify its use. Furthermore, if the user is relieved of the burden of analyzing the system states, he need not be experienced in the art of reliability computation. This opens up the use of the program as a design tool to a larger base of users. Perhaps most important, it could help to eliminate human error from the usually tedious PMS reliability calculation task.

On the other hand, referring to our example again, it is sufficient to supply a human being the following brief statement for him to accomplish the task of reliability computation: "at least 1 P and at least 2 M's and at least 1 T.d need to be functional". He then proceeds to use his knowledge of component behavior. He deduces that certain auxiliary components beyond those explicitly specified will need to succeed in order to create a system success state. For instance, if P.1, M.1, M.2 and T.d.1, in being functional, are to be part of a system success state, then S.1, K.m.1, K.m.2, and K.d.1 will additionally need to succeed.

Most important, it appears that human beings are guided principally by the Communication Axiom (see Section 2.1) in this process of deduction, in a large majority of cases. In other

words, for P.1, M.1, M.2 and T.d.1 to be part of a successful system there must be some pathways between them for information flow.[2] This point was made earlier in Section 2.1 where the distinction was also drawn between critical components and auxiliary components. The notion is that given the operational requirements on the system in terms of the critical components in it, the information provided by the manner in which system components are interconnected is sufficient in a large number of cases to deduce what subsets of the auxiliary components are necessary for each functional system state.

Modeling the style of the requirements input after what would be expected by a human being, a modified Boolean expression form was chosen. The primitives in the expression are operated upon by the standard logical AND and OR operators with the former having precedence over the latter (modifiable with parentheses of course). The primitives are of the form "at least N of X" where N is integral and $N \geq 1$. "X" is the name of a previously declared component type. This is taken to mean "at least N components of type X should be functional". There are of course two other possible forms for the primitives, namely "Exactly N of X" and "At most N of X". However, both of these, if allowed, lead to the conclusion that the system will fail if N + 1 components of type X are functional. This implies that the system is a non-coherent[3] one and, therefore, out of our purview as unlikely to be rationally designed.

We shall refer to primitives such as "N of X" as Atomic Requirements and Boolean combinations of them will be termed Compound Requirements. Atomic requirements such as "N of X" will be represented by the symbol $\psi(N,X)$. Within an atomic requirement $\psi(N,X)$, N will be termed the Integer Requirement and X will be termed the Required Component Type or simply the Required Type. The simple grammar for compound requirements is shown below:

<requirements-expression>
:: = <conjunction> | <conjunction> OR <requirements-expression>

<conjunction> :: = <primitive> | <primitive> AND <conjunction> | (<requirements-expression>)

<primitive> :: = <integer> OF <typename>

---

[2] We reiterate here that we are not considering "systems" which are assemblages of completely independent subsystems with no information flow between them. Such "systems" are easily decomposable to the case under consideration.

[3] We use this term as defined by [Barlow 75a] i.e. the structure function of the system is not monotone.

As alluded to in Section 2.1 there are three further forms of requirements input which supplement the Boolean function and provide further constraints on the evaluation thus allowing a larger space of PMS structures to be handled. These *ad hoc* side constraints strive to include semantics of individual component behavior (of which there are none built into the program) as completely and as generally as possible in the context of PMS structures. We shall consider them in turn next, however, their full impact as well as their input syntax will be clarified later (Chapter 6 and Chapter 7).

1. Internal Port Connections: Components in the PMS structure under study are represented by vertices in the interconnection graph. Arcs impinging upon a vertex correspond to the connection ports of the component represented by it. Since we are considering non-directed graphs, each arc implies that its corresponding port could potentially be an input as well as output port for the component. However, regardless of this, *within* the component, information that has entered through any particular port may, after processing, leave through one or more of the remaining ports. This internal relationship of ports in some component, say C, may be a significant aid in discovering whether two critical components, say A and B, may communicate through C. This is necessary in correctly assessing whether the Communication Axiom is satisfied by an MCRS. The default assumption in the ADVISER program in the absence of component semantics is that information potentially flows from any port to any other port *inside* any component. However, this is clearly not true, for instance, in the case of a line-printer controller. The latter will not usually act also as a conduit for information between two processors connected to it. The input to the program describing this constraint upon a particular system component is conveyed by means of a connectivity matrix of port "connections" within the component.

2. Intra Component-type Communication: In the blandest form of the model, since no component semantics are included, the Communication Axiom leads to finding K-edges between all pairs of critical components in any MCRS. However, there are many cases when information never passes between two components of the same type. For example, memories are passive components and usually never communicate with each other. When such passive behavior is to be taken into account, the use of the Communication Axiom must be modified if we are not to evaluate a pessimistic system reliability due to having unnecessarily assumed that some irrelevant inter-component paths needed to be functional. The default assumption in this instance is that critical components of *like* type never actively communicate information whereas critical components of *unlike* type will always need to communicate. The extra "constraint" being considered in this paragraph gives the user of ADVISER the ability to relax this default assumption in the case of selected critical component types. The choice of this default was not entirely arbitrary. Passive types of components such as memories of various sorts, Input/Output transducers and buses usually outnumber active types of components such as processors and device controllers in a typical PMS structure.

3. Critical Component Clusters: The third type of side constraint on the model considers the following phenomenon. In certain PMS structures, in order to have a functional system it is not sufficient just to satisfy the lower bounds on the

number of critical components of each critical component type (CCT). In addition, these functioning critical components need to satisfy criteria regarding how they are dispersed in the structure. The situation is best explained through an example. For instance, consider Figure 2-4 which depicts a multiple processor system with an inter-processor bus. Let us assume that the processors do not share the same address space. Then, for any processor to be useful when functional, at least some of its associated memory must be functional. Thus, if the minimal requirements for the PMS structure in the figure are

$$\psi(2,P) \text{ AND } \psi(4,M)$$

then the MCRS (P_A,P_B,M_E,M_F,M_G,M_H) should not be part of a system success state. This kind of behavior is observable in multiprocessor systems such as PLURIBUS [Ornstein 75]. This situation can be viewed as an association or *clustering of CCTs in substructures* of the system. In other words, if the CCTs A and B are associated or "clustered" in this fashion, then in order for any functioning components of type A in a given substructure to play a useful role, components of type B must also be functional in that same substructure. We shall, therefore, refer to a <u>cluster</u> of critical component types which are related in this manner.

The notion of clustering of CCTs is further refined in the following manner. In the general case it is not sufficient to just cluster CCTs and satisfy minimal requirements for system success. Some lower bounds are usually in force on the number of components of each such clustered type which are to be functional in a specific substructure. The bounds effectively derive from sets of inequalities which relate the number of functioning components of various CCTs. Therefore, for this cluster of CCTs, we may have, in addition, the following inequalities:

$$\text{Number of P} \geq 1 \qquad (2.1)$$

$$\text{Number of M} \geq 2 * \text{Number of P}$$

Thus for instance in Figure 2-4 (P_A,M_A,P_B,M_E,M_F,M_G) may <u>not</u> be a system success state, even though the clustering constraint is satisfied, because it may be necessary to have at least two local M's functional per functioning P to achieve system success e.g. a processor may need a minimum of, say, 8K of local memory for success and each M is a 4K board. Thus (P_A,M_A,M_B,P_B,M_E,M_F) is an MCRS which might be part of a system success state. This phenomenon of inequality relationships on the number of functioning components belonging to a set of clustered types in a substructure will be termed <u>bounded clustering</u> of critical component types. A cluster constraint to the program will consist of a set of CCTs and a set of inequalities which relate the number of functioning components of each CCT in the cluster, as in Equation (2.1) above.

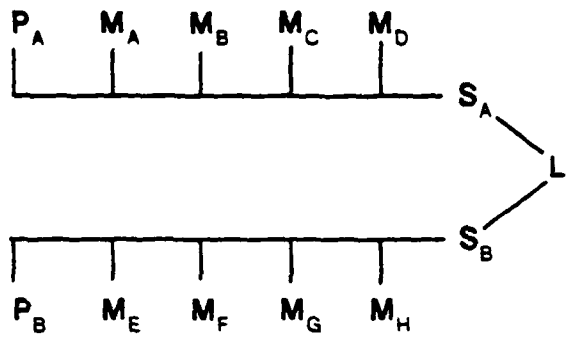


Figure 2-4: Example of a PMS structure in which clustering of CCTs occurs.

2.2.2 Program Algorithms

2.2.2.1 Detection of symmetries in the PMS *interconnection graph*

Once the various inputs have been supplied, the program may be asked to compute the reliability function. Its first act in doing so is to attempt to detect symmetric substructures, if any, within the given PMS structure. The motivation for this, as noted earlier, is to explore the resulting possibility of avoiding needless duplication of effort.

The symmetry detection proceeds by assigning vertices of the interconnection graph into equivalence classes in three steps based on three equivalence relations as follows (see Chapter 4 for details):

- Step 1: All vertices representing components of like type are assigned to the same equivalence class. Upon completion of this step there will be as many equivalence classes as there are distinct component types, say T.

- Step 2: Each equivalence class generated in Step 1 is split into further equivalence classes based on the equal-degree relation, i.e. two vertices fall into the same class iff they have the same number of arcs impinging on them. At the end of this step, the maximum number of equivalence classes present will be at most $T*d_{max}$, where d_{max} is the maximum degree of any vertex in the graph.

- Step 3: The Neighbor Class Equivalence Relation (NCER) [Gaschnig 77] is next applied to the classes resulting in Step 2 to finally detect symmetric subgraphs. The NCER is elaborated on in Chapter 4. For the moment we shall roughly

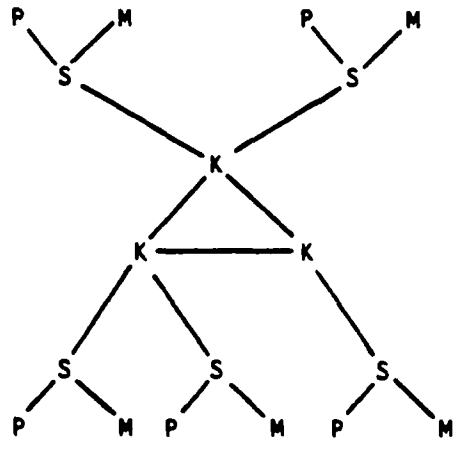
describe the nature of the NCER relation. Two vertices will be equivalenced by the NCER iff their neighboring vertices are equal in number and their corresponding neighbor vertices fall correspondingly into the same set of equivalence classes based on the NCER. At the end of this step there will be at most N NCER classes. Here N is the number of vertices in the PMS interconnection graph. This upper bound, N, on the number of classes generated by the NCER will occur in the extreme case that there is no structural symmetry in the graph and each component is of a distinct type.

The end result of this symmetry detection process is a set of equivalence classes into which the vertex set V, of the PMS graph, G(V,E), is partitioned. Each class is related to other classes in a connectivity sense that derives from the symmetric connection of the vertices in that class to their neighbor vertices in their corresponding equivalence classes. The latter are, therefore, neighbors of the former class. Thus, these equivalence or neighbor classes and their connectivity relationships may be viewed as defining another graph called the Neighbor Class Graph (NCG), G′(V′,E′). The members of the vertex set V′ of the NCG correspond uniquely to the equivalence classes on V by virtue of the NCER relation. The edges in the set E′ map the connectivity of the vertices in V by the edges in E to the connectivity of the equivalence classes that those vertices comprise. Unlike the basic non-directed graph, without self-loops, which was taken to be the model for G, G′ may have vertices in V′ which have self-loops on them. This would be the result of a case in which vertices in the same equivalence class are connected to *each other* in some symmetric fashion, thus making the equivalence class its own neighbor. Figure 2-5 shows the effect of applying the symmetry detection algorithm to an example PMS structure. This example will be described in greater detail and more will be said about NCG's in Chapter 4.

2.2.2.2 Segmenting of the PMS graph

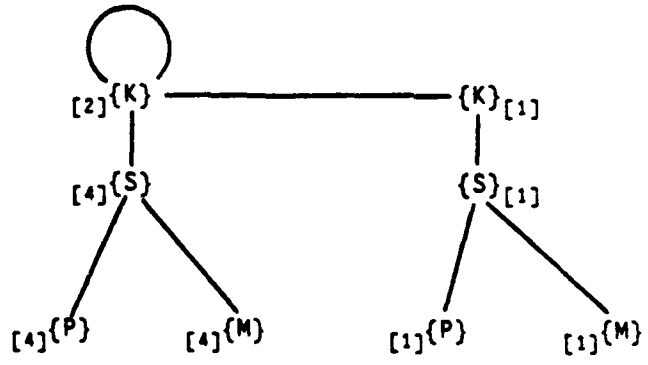
Having detected symmetries in the PMS graph the next step taken by the program is to investigate whether it is possible to segment the original PMS interconnection graph. If this is feasible then a divide-and-conquer approach may be applicable. The segmenting[4] proceeds by searching for what are termed Pendant Tree Subgraphs (PTS). These are maximal rooted tree subgraphs of the PMS interconnection graph. Their roots are articulation vertices of the graph. Furthermore, the simple path between any pair of vertices in these tree subgraphs is the only path between those vertices in the overall interconnection graph, G. It is common to find PTSs in most PMS structures. In particular, input/output subsystems typically assume this character, as in the examples of Figure 2-6.

[4]We prefer the term "segment of G" rather than "partition of G" since the latter implies the subdivision of the vertex set induced by an equivalence relation.



G(V,E)

SYMMETRY DETECTION

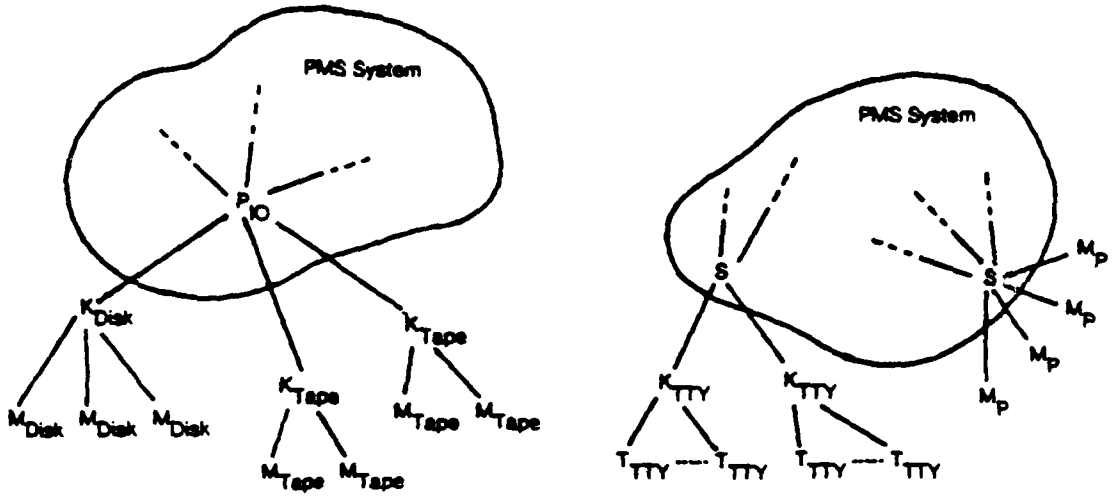


TNCG = G'(V ,E')

Key

P,M,K,S	-->	Component Types
$\{K\}_{[2]}$	-->	Equivalence Class of components of type K with cardinality of two.

Figure 2-5: Effect of applying symmetry detection algorithm to an example PMS structure. For details of this particular case see Page 83.



(a) (b)

Figure 2-6: Examples of Pendant Tree Subgraphs.

If arcs and vertices of such pendant trees, excluding their roots, are removed from the main PMS interconnection graph G, then the remaining vertices and arcs form a subgraph of G that is not tree-connected i.e. contains cycles[5]. This will be referred to as the Kernel or Core. The root vertices of the PTSs are termed interface vertices by virtue of their task of serving as communication "gateways" between components in the PTSs and the components in the Kernel. The root of each PTS has dual status as member of the PTS as well as the Kernel. These interface vertices are accorded special treatment in the reliability calculation process in view of this dual status.

[5]This is not strictly true since the outcome depends on the criterion for maximality of the pendant tree when the entire PMS graph G is a tree and thus has no cycles. See Sections 5.3 and 7.3.1.

The PTSs along with the Kernel form a natural set of segments of G on the basis of which the reliability computation task may be divided. The choice of this segmenting scheme was motivated by the earlier development, during the course of this research, of an algorithm for computing the reliability functions for PTSs (see Chapter 5). *However, the scheme for making use of these segments (see Chapter 6) does not depend on the segments being composed entirely of PTSs and the Kernel. The development of special techniques for subgraphs of G which are other than PTSs would allow an even finer segmenting of the graph without affecting the algorithms which make higher level decisions in regard to the use of these segments.*

The program discovers the PTSs in a given PMS structure G(V,E) by starting with those leaf vertices of G′ which represent classes of leaf vertices of G[6]. These "germinal trees" are then "grown" upward towards the root by adding on neighboring vertices of these leaves and at each step merging the germinal trees which overlap. This process continues (subject to termination conditions described in Chapter 6) until no more adding of vertices or merging of trees is possible. At this point a set of tree subgraphs of G′ have been generated. Each of these trees in G′ may represent one PTS of G or a set of PTSs. In the latter instance all PTSs in the set will be symmetric.

2.2.3 The OVERLORD routine

2.2.3.1 Generation of feasible MCRSs

The OVERLORD routine in ADVISER is the heart of the program. In this routine critical components are "drawn" from the various segments in various ways to try to satisfy the various requirements. Each "draw" is then checked to see that requirements on G and other side-constraints (Section 2.2.1.3) as well as the Communication Axiom are satisfied. The partial results of each successful draw are stored away in a special data-structure. At the end of the drawing process the partial results are retrieved and merged to provide the system reliability function.

The sketchy explanation above may be clarified by evoking an analogy to drawing colored balls from urns. Balls are analogous to critical components and urns to the segments of G. The colors of the balls represent the various critical component types. Each urn contains a

---

[6] It is possible for leaf vertices of G′ to represent classes of vertices of G which are not leaves of G. See Section 4.5.

certain number (possibly zero) of balls of each color. The requirements may then be rephrased as the desire to choose balls from urns in such a way as to satisfy a minimum on the total number of balls of each color that are chosen. This is further subject to side-constraints such as (i) if some balls of color A are chosen from urn X then some balls of color B must also be chosen from X or else neither. (Clustering of component types) or (ii) if colors A and B are to be simultaneously chosen from urn X then a minimum of m balls of color A and n balls of color B must be chosen from the urn (Bounded clustering of component types).

We may now examine the process of making a "draw". Let us consider the simple case of a system where G has been segmented into five segments (urns) P_1 through P_5. Let us assume also that the only (atomic) requirement is $\psi(4,t)$. Furthermore, let us assume for a while that each of the urns contains 4 or more balls of color t. Then the draw proceeds by generating the 5-compositions of the integer 4 as in Figure 2-7(a).[7]

Each integer-part of each 5-composition represents the number of components of type t drawn form the corresponding segment of G. Since the preliminary assumption for the purpose of the exposition was that each segment contained at least four t's all the 5-compositions in this instance represent draws that are feasible. Once a draw is feasible, it may be made and represents one possible alternative for satisfying the requirements. Of course, the side-constraints and the Communication Axiom must be satisfied before the MCRS, so drawn, constitutes part of a functioning system.

In general, not all of the segments of G will contain enough components of a given type to support a given atomic requirement on that type. Figure 2-7(b) depicts an example of such a case. In this instance, against the requirement of four t's, none of the segments P_1 through P_5 are able to supply *all* four, and P_2 and P_5 contain no t's at all. These upper bounds on the number of components of type t which may be drawn from a particular segment can, in general, drastically curtail the number of draws that are feasible. The program actually generates all possibilities when the compositions of an integer are desired, testing each one

---

[7] A composition of the integer m into n parts, that is an n-composition of m, is a representation of the form

$$m = k_1 + k_2 + ... + k_n,\ k_i \geq 0,\ i = 1,2....,n$$

with regard to the particular order of the k_i's. Thus, there are exactly four 2-compositions of the the integer 3, namely, $3+0$, $2+1$, $1+2$ and $0+3$. In general there are $\binom{m+n-1}{n-1}$ n-compositions of the integer m (for a derivation of this see [Liu 68]). The n-compositions are not to be confused with n-*partitions* of the integer m. The latter take the same form as above except that $k_i > 0$, $i = 1.....n$ <u>without</u> regard to the order of the k_i's. Thus there are only two 2-partitions of the integer 3, namely, $3+0$ and $1+2$.

		Graph Segments				
		P_1	P_2	P_3	P_4	P_5
	1.	4	0	0	0	0
	2.	3	1	0	0	0
5-compositions	3.	3	0	1	0	0
(i.e. draws)	4.	3	0	0	1	0
	5.	3	0	0	0	1
	6.	2	2	0	0	0
	7.	2	1	1	0	0
	8.	2	1	0	1	0
		...	...	...	...	...
		...	...	...	...	...
		...	...	...	...	...
$\binom{4+5-1}{5-1} = 210.$		0	0	0	0	4

Note: all graph segments here have four or more t's.
Hence, all compositions in this table are feasible.

(a)

		Graph Segments (max. t's available in segment, within parentheses)					
		P_1 (3)	P_2 (0)	P_3 (2)	P_4 (1)	P_5 (0)	
	1.	4	0	0	0	0	(infeasible)
	2.	3	1	0	0	0	(infeasible)
5-compositions	3.	3	0	1	0	0	(feasible)
(i.e. draws)	4.	3	0	0	1	0	(feasible)
	5.	3	0	0	0	1	(infeasible)
	6.	2	2	0	0	0	(infeasible)
	7.	2	1	1	0	0	(infeasible)
	8.	2	1	0	1	0	(infeasible)
		...	...	...	...	...	
		...	...	...	...	...	
		...	...	...	...	...	
$\binom{4+5-1}{5-1} = 210.$		0	0	0	0	4	(infeasible)

(b)

Figure 2-7: An example of drawing critical components from segments of G.

against the upper bounds dictated by the contents of each segment for the particular case.[8] However, only those possibilities which represent the feasible draws emerge from the generator function. This was deemed acceptable, as the overhead is very small compared to the computing requirements of other portions of the program.

Thus far in this section we have only considered atomic requirements of the type $\psi(m,t)$. In the more usual case the requirements will consist of a boolean expression on such atoms. These expressions may be naturally divided into two classes: those that contain only conjunctions of atomic requirements (Conjunctive Requirements) and those that contain at least one disjunction in addition to possibly containing conjunctions (Disjunctive Requirements). Let us focus attention on the former class briefly.

In the worst case all the segments of G contain enough components of each critical component type to satisfy each and every atom in the conjunctive requirement. In other words, all possible draws will be feasible. Then the total number of feasible draws over the entire conjunctive requirement

$$\Lambda_{i=1}^{r}\ \psi(m_i,t_i)$$

is given by

$$f_c = \Pi_{i=1}^{r} \binom{m_i + n-1}{n-1}$$

where n is the number of segments of G and each segment contains at least m_i components of type t_i. In the purely disjunctive case (no conjunctions in the overall requirement) the requirement is

$$V_{i=1}^{r}\ \psi(m_i,t_i)$$

and the number of feasible draws is

$$f_d = \Sigma_{i=1}^{r} \binom{m_i + n-1}{n-1}$$

The numbers f_c and f_d represent the upper bounds on the number of cases to be analyzed in the case of purely conjunctive and purely disjunctive requirements respectively. However, in the more usual case of a mixture of conjunctions and disjunctions, the worst case bound can

---

[8] The algorithm used by the ADVISER program to generate feasible compositions is an adaptation of the NEXTCOM algorithm in [Nijenhuis 78]. See also Chapter 6.

be much higher. A closed form solution for the worst case bound for this intermediate variety of requirement expressions is unobtainable since it depends on the values of the m_i's and n, as well as the positions and precedence order of the conjunctions and disjunctions in the expression.

The generation of all possible sets of compositions for a conjunctive requirement is a backtrack procedure which uses a stack discipline. This may be observed by considering the following example of a conjunctive requirement:

$$\psi(m_1,t_1) \wedge \psi(m_2,t_2) \wedge \psi(m_3,t_3) \tag{2.2}$$

Let us assume there are n segments of G. Then for each n-composition of m_1 *all* n-compositions of m_2 have to be generated. In turn for each of the latter, *all* n-compositions of m_3 have to be generated. This may be done in a systematic manner for the general case of a conjunctive requirement by the algorithm in Figure 2-8. The action of the algorithm is analogous to the operation of an odometer. For instance in the example of the requirements expression (2.2) above, the atoms are analogous to the wheels of the odometer. With the given ordering of the atomic requirements in the expression (the order is not of concern due to the commutativity of $\wedge$) $\psi(m_3,t_3)$ can be thought of as the fastest moving wheel of the odometer and $\psi(m_1,t_1)$ the slowest moving wheel. Each composition of the requirement integer m_3 over the n segments of G is analogous to a digit on the fastest wheel. Therefore, in the algorithm the next n-composition of m_2 (digit on the wheel) is generated (wheel corresponding to $\psi(m_2,t_2)$ is advanced one position) only when all n-compositions of m_3 are exhausted. Upon the generation of the next n-composition of m_2, the sequence of n-compositions of m_3 are sequentially generated once more, and so on. The process terminates in the case of our example when all n-compositions of m_1 have been exhausted.

Returning to the stack discipline for generating all possible sets of n-compositions, we may liken each level of the stack to an odometer wheel. All the compositions at level r + 1 of the stack are generated before the stack is "popped" one level and the next composition at level r is generated. Following this the stack is once again "pushed" to level r + 1 to generate the next cycle of n-compositions at that level.

It is apparent that in the worst case, at each level, k, of the stack all the n-compositions m_k are generated repeatedly in sequence. The number of times this sequence is repeated at that level is given by

```
begin integer r; composition array cstack[S];
          Comment S=total number of atoms in conjunctive requirement
            and each location of array cstack holds one n-composition
            of an integer;
r←0; cstack[1 thru S]←0;

loopa:
repeat

   if r < S
   then

      r←r+1;
      cstack[r]←(first n-composition of m_r);
                     Comment i.e. (m_r,0,0,....,0);

   else

       loopb:
       repeat

          (use contents of array cstack as the next draw);
          if
              (next n-composition of m_S cannot be generated)
            Comment the last n-composition of m_r is (0,0,....,m_r);
          then
              r←r-1;
              leave loopb
          else
              cstack[S]←(next n-composition of m_S)
          fi
       endrepeat
   fi;

   until (it is possible to generate next n-composition of m_r)
   do
       if (r←r-1) = 0 then leave loopa fi;
              Comment until no more compositions to generate;
   od;

   cstack[r]←(next n-composition of m_r)

endrepeat

end;
```

Figure 2-8: Algorithm to generate all possible combinations of n-compositions.

$$\prod_{r=1}^{k-1} \binom{m_r + n-1}{n-1}$$

2.2.3.2 Satisfying the Communication Axiom

Each of the feasible draws generated constitutes an MCRS which may or may not be part of a functional system state. This depends, of course, on whether side-constraints have been met and the Communication Axiom has been satisfied. Checks are made with regard to these by the OVERLORD routine. For a given feasible draw (MCRS) the critical components chosen in that draw will be scattered in some fashion among the segments. In order to satisfy the Communication Axiom the critical components in the pendant tree segments will have to communicate with each other and to critical components drawn from the Kernel, through paths in the Kernel. Moreover, information may flow in and out of the Kernel only through the root vertices of the pendant trees, since these are articulation vertices of G.

Thus the question whether the Communication Axiom may possibly be satisfied by a given candidate draw may be separated into two concerns, namely:

1. Critical components in tree segments should be able to communicate with the component represented by the root vertex of that tree, and

2. The root vertices, of the tree segments that contain the critical components of this candidate draw, should be able to communicate with each other and critical components drawn from the Kernel, via paths in the Kernel.

The former concern is addressed by the algorithm TREEREL developed for the PTSs (see Chapter 5). The latter concern is the domain of the OVERLORD routine and is addressed in Chapter 6.

For each draw, therefore, the OVERLORD routine performs checks on the Kernel. For each such iteration, depending on which segments the critical components are drawn from, the set of relevant root vertices (and therefore their respective PTSs) and the set of critical components drawn from the Kernel is subject to change. This change is unpredictable and depends on the requirement expression, the scattering of the available critical components in various portions of the system and the nature of the side-constraints. It might appear, therefore, that large amounts of computation e.g. deriving partial results for the pendant trees etc. might have to be invested at each iteration. However, there are some unchanging aspects of the situation which the program can use to good effect and so it does.

The reader will have noticed when the drawing process was described earlier that for any level r in the stack the n-compositions of m_r were generated repeatedly except in the case of

$r = 1$. The program is thus able to anticipate that certain partial results will be needed in several iterations. Such partial results are computed once initially and stored away in special hash tables. In general, in a compound requirement, many atoms in the expressions may refer to the same critical component type, say t. Let one such atom be $\psi(m_i,t)$. When compositions of the various m_i's are taken over the segments of G, the minimum number of critical components of type t that may be drawn from some segment, say p, is one. The maximum number of critical components of type t, say m_{max}, which may be expected to be drawn from p is the lesser of (i) the number of components of type t in p and (ii) the largest m_i in any atom of the form $\psi(m_i,t)$ in the compound requirement. Thus, the OVERLORD routine generates partial results for each such segment p for the set of atomic requirements $\{\psi(j,t)\}$ where $j = 1,....,m_{max}$. This is done for each critical component type. These stored partial results are then later retrieved and used during the process of generating feasible compositions.

The Kernel is treated slightly differently, though even here such reusing of intermediate results is possible; they are just of a different nature. The OVERLORD routine checks for the existence of K-edges through the Kernel which lead among the pendant trees (or what is equivalent, their root or interface vertices) and other critical components drawn from the Kernel. Thus, in this case, it generates and stores away partial results for such K-edges between all possible pairs of critical components and/or interface vertices in the Kernel.

In summary, all partial results which could possibly be used in the computation are generated once in the beginning and stored away in hash tables. For each iteration, then, the OVERLORD routine retrieves and uses the appropriate partial results after ascertaining that the draw for that particular iteration will satisfy the various side-constraints and the Communication Axiom. The methods of representation and combination of the partial results alluded to above are the subject of the next section.

2.2.3.3 Representation of Reliability Expressions

At all stages of the computation of the reliability function, the identity of each component in the structure is retained in the reliability expressions which are the partial results and the final reliability function. As a consequence, recalling that statistical independence of component failure behavior has been assumed, the structure of the partial results and the final reliability function will be very similar to that of a Boolean function in its minterm canonical form. Each partial result will be a function of the reliabilities of some subset of the system components. The expression which is the body of the function will consist of "minterms". Each minterm will consist of the algebraic product of the probabilities of success (reliabilities) of a subset of

components. Each of these factors of a minterm will appear only once in the minterm and will be raised to the unit power. Each minterm will, in addition, be prefixed with a positive or negative sign. We shall term such an expression a Canonical Reliability Polynomial. An example of one is given below:

$$R_X = R_1 + R_2 + R_3 - R_1R_2 - R_2R_3 - R_1R_3 + R_1R_2R_3$$

R_X is the system reliability for a 1-out-of-3 system. Such a system is functional only if at least one of its three components is functional. R_1,R_2 and R_3 are, respectively, the reliabilities of the three components. The above function could have been reduced to a form that was not canonical if, say, $R_1 = R_2 = R$, whereupon:

$$R_X = 2R + R_3 - 2R^2 - R_3^2 + R^2R_3, \quad R_1 = R_2 = R$$

However, the canonical form is the most general and represents the reliability of a system wherein no two components have identical reliability functions. All non-canonical forms may be derived from the canonical form by appropriate algebraic substitution, although the reverse is not possible in general.

This, then, is one of two primary motives for retaining partial results in canonical form. In other words, the fact that two or more components in the system may have identical reliability functions does not change the canonical form since its only proviso is that the components have statistically independent failure behavior.

The other equally important motive for retention of canonical form concerns the robustness and simplicity of the algorithm to combine the partial results in conjunction or disjunction. The algorithm and its data structures are the subject of Chapter 3. It will suffice here to note two simple points.

Firstly, factors in the minterms of a canonical reliability function are always raised to the unit power and are never replicated within the minterm. Hence each minterm may be represented by a string of (N + 1) bits (N system components + 1 sign bit)[9] wherein each of the first N bits represents a unique factor (component). Furthermore, a canonical reliability function may then be represented as an unordered list of bitstrings, each bitstring in the list representing one minterm.

---

[9] This idea has been utilized before though not quite in the same fashion. See [Gandhi 72].

Secondly, operations on pairs of such lists will be composed of simple logical operations on pairs of bitstrings from the two lists. Thus the resultant list will contain bitstrings arising from the Cartesian product of bitstrings from the two input lists using those logical operations. These logical operations on bitstrings are available as hardware instructions on most computers.

A price is paid, however, for the use of the canonical form of the reliability function since the number of terms in the canonical form usually exceeds those in a simplified form for more complicated problems. Indeed, it is in the code that processes these lists of bitstrings where the ADVISER program spends much of its computation time (see Chapter 7). This problem is partially averted by assigning bits in the bitstrings for *partial results which are statistically independent*. This results in smaller lists in the canonical form. The partial results which are allotted bits in the bitstrings eventually become separate numeric calculations, the results of which are substituted into the main reliability function when it is numerically evaluated.

2.2.4 Program Output

2.2.4.1 Printing of Results

The final stage in the operation of the program consists of reducing the canonical form of the system reliability function which was generated and then printing it out appropriately.

The reduction proceeds by noticing, from the component-type and interconnection graph declarations, which components are of the same type (i.e. have identical reliability functions). Appropriate substitutions and algebraic simplification are then performed to obtain the reduced function. The simplifications are rudimentary and limited to cancellation of like terms of opposite sign and the gathering of like terms of like sign. Factoring is not attempted, for example. The resulting non-canonical symbolic form is then a function of the symbolic reliabilities of the *component types* and all individual component identities are lost as a result of the reduction. The program does, however, keep a copy of the canonical form should several printings be desired. The output of the program in its current state of development is a text file. This file will, upon the user's option, contain the text of either a SAIL [Reiser 76] program procedure, or a FORTRAN function, which computes the system reliability function $R_{sys}(t)$ for the input PMS structure under the input requirements. The procedure or function will have as a single parameter the time t at which the reliability is to be computed. The file may then be compiled with the appropriate compiler and used in numerical calculations using the generated reliability function. The advantage of having the system reliability output as a

program is that different reliabilities may be used for the individual component types by editing the program rather than redoing the ADVISER computation. As a third option ADVISER will generate an output file containing the partial result and system reliability polynomials in a form suitable for input to the MACSYMA symbolic manipulation program [Macsyma 77]. This makes it possible to carry out more advanced manipulations such as factoring, symbolic differentiation etc. on the system reliability function.

2.3 Conclusion

This completes the overview of the ADVISER program and lays the foundation for more detail on each of the computation phases to be described in the ensuing chapters. The reader is urged to refer back to this overview to obtain a general context within which to understand the more detailed discussions which are contained therein.

Chapter 3
Intermediate Representations

3.1 Introduction

The reliability of a system of components is a composite of many factors. The individual component reliabilities are, naturally, important. However, so is the manner of physical or logical interconnection of the components which comprise the system. The functional behavior of an individual component may also contribute toward determining the system reliability. Traditionally, a few methods have commonly been used to represent the interdependence of all these contributing factors in a form that is amenable to the use of formal methods to compute the reliability of the system from that of its component parts. The methods in common use are generally based in graph theory and make use of the extensive results derived in that field. Some frequently used methods will be briefly described and contrasted in this chapter. The discussion will serve as the basis for the choice of a particular representation, the Series-Parallel connected Reliability Block Diagram (SPRBD), or series-parallel two-terminal network, for use in ADVISER. Subsequent sections will examine simple algorithms and data structure for employing this representation to generate reliability functions in symbolic form. Figure 3-1 shows the place, within the overall scheme of ADVISER, of the SPRBD algorithm package to be discussed in this chapter.

3.2 Some commonly used representations

As has been pointed out in Chapter 2 three basic classes of data are required to generate the reliability measures for a system, namely

1. The physical or logical interconnection structure of the system
2. The individual reliabilities of the components comprising the system.
3. The requirements or constraints on the system, in terms of sets of working components, that define under what conditions the system is considered to be operational.

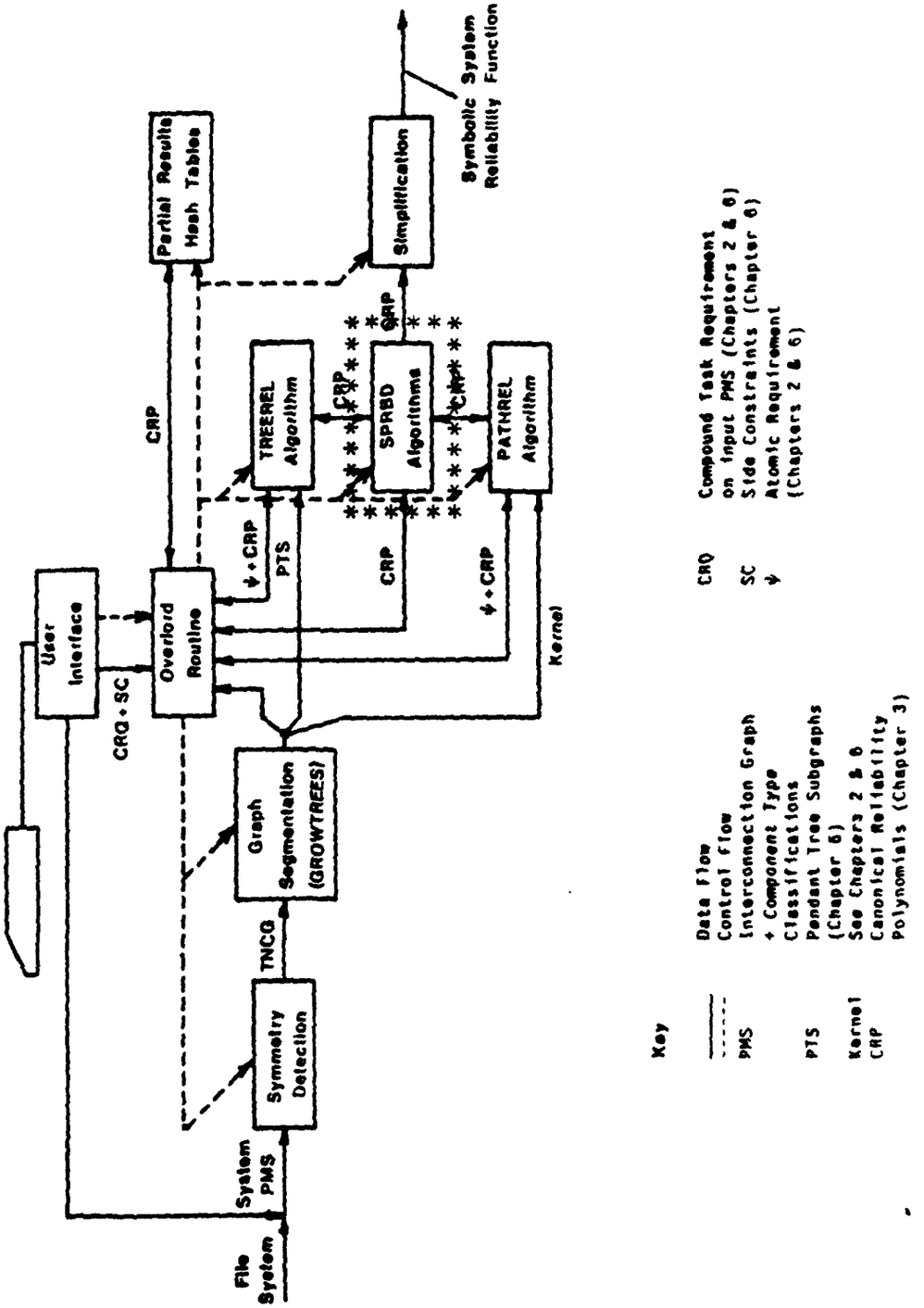


Figure 3-1: The portion of the ADVISER structure discussed in Chapter 3.
Also see Page 18.

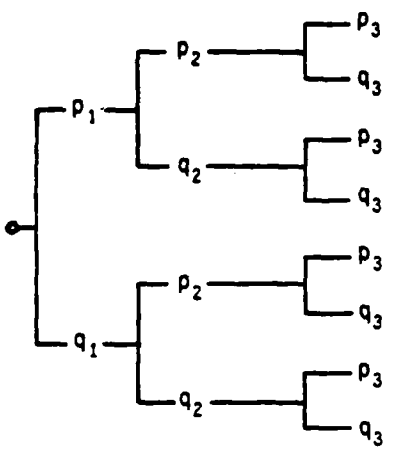
Thus far in practice the general paradigm of reliability computation has been as follows. The human being uses the physical interconnection structure of the system and the operational requirements on it, to generate a graph-theoretic data representation which embodies the system's reliability characteristics. This representation is then typically processed by a computer program which eventually computes the relevant reliability measure for the system. We shall call such a data representation an Intermediate Representation. In this chapter we shall be concerned with two basic types of Intermediate Representations, namely Fault Trees and Reliability Graphs. A large part of the published literature, however, also deals with a third type of representation, the network. A graph, G(V,E) is used to model, say, a computer communication network wherein the homogeneous vertices {V} of the graph represent computers and the homogeneous arcs {E} represent the communication links between them. Reliability concerns are then of the type "What is the probability that any two vertices of the graph are able to communicate at any time?" or "If the links are subject to stochastic failures what is the probability that enough links will be operational at any time to preserve at least a spanning tree of the network?", and so on. We shall leave a consideration of such problems to a later chapter since although the model is an idealization it still closely resembles the actual system interconnection structure and is thus further removed from being an Intermediate Representation.

3.2.1 Probability Trees

According to two of the basic assumptions made in Chapter 2, failure processes in components are s-independent and each component may be in one of two states, failed or working. Thus a system consisting of n components will occupy one of the states $S_i \in \{S\}$, $1 \leq i \leq 2^n$. Of these 2^n states some subset {F} will consist of the failed states of the structure whereas the subset {W} will consist of the working states. In addition $\{W\} \cup \{F\} = \{S\}$. The reliability of the system is then expressed by

$$R_{sys}(t) = \Sigma_{\forall F_i \in \{F\}} \Pr(F_i) = 1 - \Sigma_{\forall W_j \in \{W\}} \Pr(W_j)$$

The 2^n system states may be viewed as the leaves of a binary tree n levels deep wherein each level corresponds uniquely to some one component in the system. For example, Figure 3-2 shows the probability tree for a system with $n = 3$ components which is functional iff at least two out of the three components are functional (this is a so-called 2-out-of-3 structure). The reliability function for the system may be symbolically derived from its probability tree with a simple algorithm. Assign each state (tree leaf) a unique integer i, $1 \leq i \leq 2^n$. The probability of occurrence of state i, $\Pr(S_i)$, is the product of all the probabilities (labels of vertices in the tree)



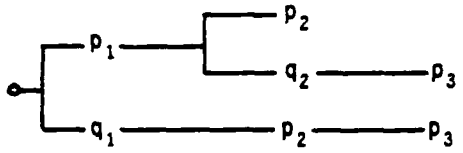
State	Disposition
123	Working
123′	Working
12′3	Working
12′3′	Failed
1′23	Working
1′23′	Failed
1′2′3	Failed
1′2′3′	Failed

Key

1 --> Component 1 working
1′ --> Component 1 failed

p_i --> Probability component i works
q_i --> Probability component i fails

(a)



State	Disposition
12X	Working
12′3	Working
1′23	Working

Key

X --> "Don't Care" state
1 --> Component 1 working
1′ --> Component 1 failed

p_i --> Probability component i works
q_i --> Probability component i fails

(b)

Figure 3-2: Probability Tree for 2-out-of-3 structure.
(a) Complete (b) Reduced.

on the path from the root of the tree to the leaf representing state i, e.g. in Figure 3-2(a) $Pr(S_3)$ = $p_1q_2p_3$. If the vertices of the probability tree are labelled with probabilities as in Figure 3-2 then it is easy to see how a factored symbolic system reliability function may be derived. A post-order traversal of the binary tree is done and the label of each node is multiplied into the algebraic sum of the symbolic results returned by the traversal of the left and right subtrees. For the leaves of the tree simply the label is returned.

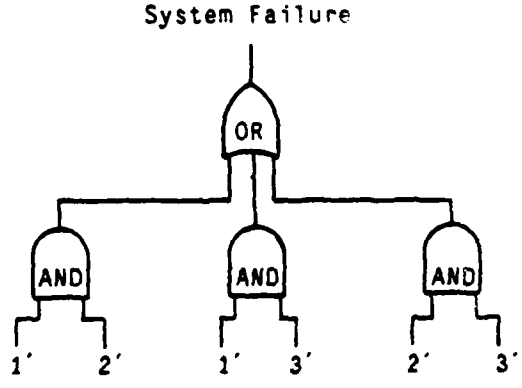
For the complete binary probability tree this simple algorithm will examine all the 2^n possible states of the system since there will be 2^n leaves in the tree. This is exactly equivalent to state enumeration by running through the binary counting sequence for n-bit integers with each bit uniquely representing a component and a binary 0 in a bit position implying component failure while a binary 1 implies component success. For each state that is examined in the worst case $O(n^2)$ examinations of pairs of functioning components need to be made. This is to ensure that, even though the required number and types of the necessary components are functional, the appropriate connectivity requirements between those components are met, for the state to be classified a functional one. Thus the algorithm will take $O(n^2 2^n)$ operations to complete in the worst case.

That there is room for more efficient use of this representation is evident from examining Figure 3-2. For instance consider the mutually exclusive states (123) and (123′) in that figure. Their associated probability function terms are $p_1p_2p_3$ and $p_1p_2q_3$ which when added produce $p_1p_2(p_3 + q_3) = p_1p_2$. In this instance the minimum requirements for system success are available by the time the descent reaches the second level of the tree so the leaves 1 and 2 need not have been generated when the tree was constructed. Component 3, in this case, is a "don't-care" component. Similarly, in the case of state (1′2′3′) it is evident by the time the second level of the tree is constructed that this state is going to be a failed system state. Thus, the tree may be pruned here. The number of leaves in the tree be reduced by such observations so that the algorithm has less leaves to examine. The pruned version of the tree in Figure 3-2(a) is shown in Figure 3-2(b). [Chelson 71] describes a method of computing system reliability which uses probability trees.

One advantage of using the probability tree representation for symbolic reliability function generation is that the algorithm will always produce a factored version of the reliability function. This is attractive when numerical computation is envisioned using the factored reliability function since the accumulation of round-off and truncation errors is reduced. The major disadvantage of the approach is its exponential time complexity which is very sensitive to the ordering imposed on the system components. This restricts the method to small problems.

3.2.2 Fault Trees

Another Intermediate Representation widely used in system reliability studies is the Fault Tree. A Fault Tree is in general an incomplete n-ary tree. There are two kinds of vertices in such a tree, namely (a) failure events and (b) Boolean operators (e.g. AND, OR, XOR, etc.). All vertices at any level of the tree are of the same type and the two kinds alternate between alternate levels of the tree. The root vertex always represents the major failure event (e.g. system failure) being studied and known, by convention, as the Top Event. The (usually) independent failure events which are the leaves of the tree are known as the Basic Events. Figure 3-3 refers to the same 2-out-of-3 structure analyzed in Figure 3-2, It may be seen that a fault-tree is equivalent to stating the top event as a boolean function of the other failure events at lower levels of the tree. An analysis of a fault tree for a system can also provide the sequence of failure events that would lead to the occurrence of the top event or system failure.



Key i' --> Component i failed

Figure 3-3: Fault Tree for 2-out-of-3 structure

Fault trees are specific instances of a more general representation called Event Trees. The latter are similar but the events are not necessarily failure events. When the basic events and the top event are failure events then the event tree is called a fault tree. Fault trees find their most extensive application in the engineering reliability analysis methodology known as FMECA (Failure Modes, Effects and Criticality Analysis). In this methodology, a fault tree is

built for the complex system being engineered. The act of construction of the fault tree forces the designer to consider all possible combinations of basic failure events. It is thus invaluable in discovering all those specific combinations of failure events which would lead to system failure. In this the fault tree, once constructed, also stands as documentation of the various possible *sequences* of component failures which will lead to system failure.

A fault tree may also be analyzed to compute the probability of the top event given the probabilities of the basic events. Since the top event is usually system failure, its probability of occurrence is the system unreliability, i.e. the probability must be subtracted from 1 to get the system reliability. Fault tree analysis usually proceeds by generating the minimal cut sets of basic events. A minimal cut set is a minimal set of basic events such that their occurrence insures system failure. A simple algorithm for generating the minimal cut sets for an event tree is given in [Barlow 75a]. In the simplest case the basic events in a cut set are statistically *independent and the minimal* cut sets are $K_1, K_2, \ldots, K_k$ for a given fault tree and are disjoint. Then the probability of the top event is given by

$$\Pr\{\text{Top Event}\} = 1 - \Pi_{s=1}^{k} (1 - \Pi_{i \in K_s} p_i)$$

where, p_i is the probability of occurrence of basic event i.

The literature on fault trees is vast but a good introduction to fault tree analysis is given in [Barlow 75b]. [Bennetts 75] gives a *procedure for computing* a minimal sum-of-products expression for the system reliability using a fault tree.

3.2.3 Reliability Graphs

The Reliability Graph is the third class of Intermediate Representation that we shall consider in this chapter. Reliability Graphs are directed graphs with one vertex of zero in-degree, or source vertex, and one vertex of zero out-degree, or sink vertex. Components, in the system whose reliability is being calculated, are associated with the set of vertices and/or the set of arcs in the reliability graph. More than one vertex (arc) of the reliability graph may represent a given component in the system. Each simple path from source to sink in the reliability graph then represents a set of system components. Each such set is one minimal set of components which, if functional, ensure the functioning of the system. This may be stated more formally as follows.

$$Ev_{P_i} = \cap_{k \in P_i} Ev_k \Rightarrow Ev_{sys} \quad 1 \leq i \leq m$$

where:

- Ev_x is the event "entity x is functional",
- P_i is the set of components represented by path i in the reliability graph from its source to its sink.
- m is the total number of simple paths in the Reliability graph from source to sink

A path P_i in the reliability graph is said to be functional iff the set of components represented by all vertices along that path are functional. Since the event Ev_{sys} will occur if at least one of the paths P_i, $1 \leq i \leq m$, is functional, we may make the following assertion regarding the probability of system success.

$$Pr(Ev_{sys}) = Pr(\cup_{i=1}^{m} Ev_{P_i})$$

By the familiar expression from basic probability theory which relates the probability of a union of events to the probabilities of the individual events we have

$$Pr(Ev_{sys}) = \Sigma_{i_1} Pr(Ev_{P_i}) - \Sigma_{i_1<i_2} Pr(Ev_{P_{i_1}} \cap Ev_{P_{i_2}}) + \Sigma_{i_1<i_2<i_3} Pr(Ev_{P_{i_1}} \cap Ev_{P_{i_2}} \cap Ev_{P_{i_3}}) + \ldots\ldots$$

$$\ldots\ldots + (-1)^{m-1} Pr(\cap_{k=1}^{m} Ev_{P_k}),$$

where $1 \leq i_1 < i_2 < i_3 \ldots\ldots \leq m$.

If the probabilities we are concerned with are reliabilities, then the LHS of the above equation becomes the system reliability. Using the assumption of s-independence of failure processes of system components from one another and the fact that

$$Pr\{Ev_{P_i}\} = Pr\{\cap_{k \in P_i} Ev_k\} = \Pi_{k \in P_i} R_k(t)$$

where $R_k(t)$ is the reliability function of component k as a function of time t, we have

$$R_{sys}(t) = \Sigma_{i_1} \Pi_{k \in P_{i_1}} R_k(t) - \Sigma_{i_1<i_2} \Pi_{k \in P_{i_1} \cup P_{i_2}} R_k(t) + \ldots\ldots + (-1)^{m-1} \Pi_{P_1 \cup P_2 \cup \ldots \cup P_m} R_k(t)$$

where $1 \leq i_1 < i_2 < i_3 \ldots\ldots \leq m$.

This result may be used as the basis for a simple algorithm to derive the reliability function of a system from its reliability graph. [Gandhi 72] describes such an algorithm. In brief, the

algorithm works as follows. The reliability of a path is the product of the reliabilities of the vertices that lie along the path in the reliability graph. The first term in the equation above is then the sum of the reliabilities of all simple paths in the reliability graph. The second term is composed of the sum of products of reliabilities of components in the unions of pairs of simple paths. Likewise the third term corresponds to unions of triples of simple paths, etc. The algorithm in [Gandhi 72] uses bit vectors with one bit assigned uniquely to each system component. Each simple path then consists of a bit vector with the bits corresponding to its components set to one. Unions of path component sets is then are then constructed by OR-ing together the corresponding bit vectors.

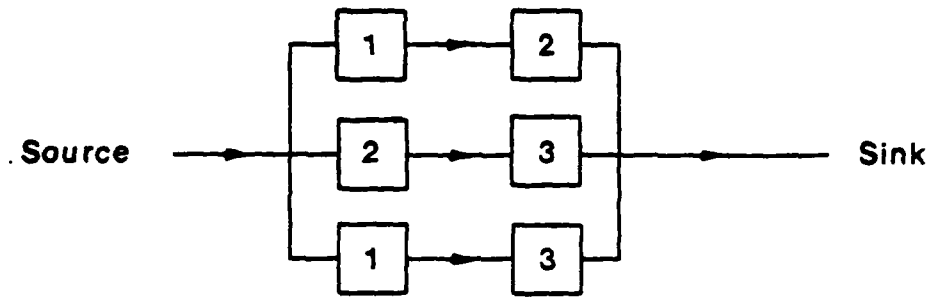
Recent work has been reported by [Satyanarayana 78] which uses the reliability graph to compute the symbolic reliability function of the system whose reliability characteristics are represented by that graph. [Aggarwal 78] describes how logical signal relations may be used to manipulate reliability graphs to obtain symbolic reliability functions. In each case, however, the reliability graph is presumed to exist and its construction is left to the designer after he has analyzed the system being studied. In the ADVISER program the work of generating the intermediate representation is assumed by the program itself which works directly with the interconnection graph of the Processor-Memory-Switch structure being analyzed. ADVISER uses a subclass of reliability graphs as a model for its intermediate representation and this is the subject of Section below.

3.2.3.1 Reliability Block Diagrams

Reliability Block Diagrams (RBDs) as a class of intermediate representations are a subset of reliability graphs. RBDs correspond to reliability graphs wherein the vertices represent system components and thus are labelled with failure probabilities. The arcs are perfectly reliable and serve only to indicate the connections between vertices. RBDs are conventionally drawn with a box to represent each vertex of the reliability graph. Figure 3-4 is one possible RBD for our running example in this chapter, the 2-out-of-3 system.

The RBD for the 2-out-of-3 system happens to be a series-parallel graph. Figure 3-5(a) is an example of an RBD which is not series-parallel. However, the following theorem shows that an RBD of any kind may be transformed into a stochastically equivalent series-parallel RBD.

> **Theorem 3.1:** A non series-parallel RBD can always be transformed into stochastically-equivalent series-parallel RBD.
>
> **Proof:** The probability of success for any RBD, G, is the probability that at least one simple path from source to sink is functional. Let P_i be the i^{th} simple path from source to sink in G; where $1 \leq i \leq m$ and m is the total number of simple paths from



Key
i --> Component i functions

Figure 3-4: Reliability Block Diagram for 2-out-of-3 structure

source to sink in G. These m paths may, or may not be composed of disjoint component sets. Construct another RBD, G′, in which there are m disjoint paths from source to sink. Furthermore, construct path i in G′ to correspond uniquely to the i^{th} simple path in G, with the same number and kinds of vertices (this is called the minimal path representation). Hence we see that

$$\Pr\{G' \text{ is successful}\} = \Pr\{U_{i=1}^{m} \{P_i \text{ is successful}\}\}$$
$$= \Pr\{G \text{ is successful}\}$$

A stochastically equivalent RBD to the one in Figure 3-5(a) is shown in Figure 3-5(b)

3.3 The Series-Parallel RBD in ADVISER

The Series-Parallel Reliability Block Diagram (SPRBD) intermediate representation was chosen for implementation in ADVISER for the following reasons:

- The RBD encodes the Boolean relationships between the successes of components in the structure which yield the success of the system. It does this without necessarily explicitly showing the intermediate success events in the system. This achieves economy of space over the equivalent fault tree representation ([Shooman 70] shows in an informal fashion that fault trees and reliability block diagrams are equivalent in information content). Moreover, unlike the probability tree the reliability block diagram does not show all possible success states of the system.

- The SPRBD is nicely recursive in structure and only two kinds of operations are

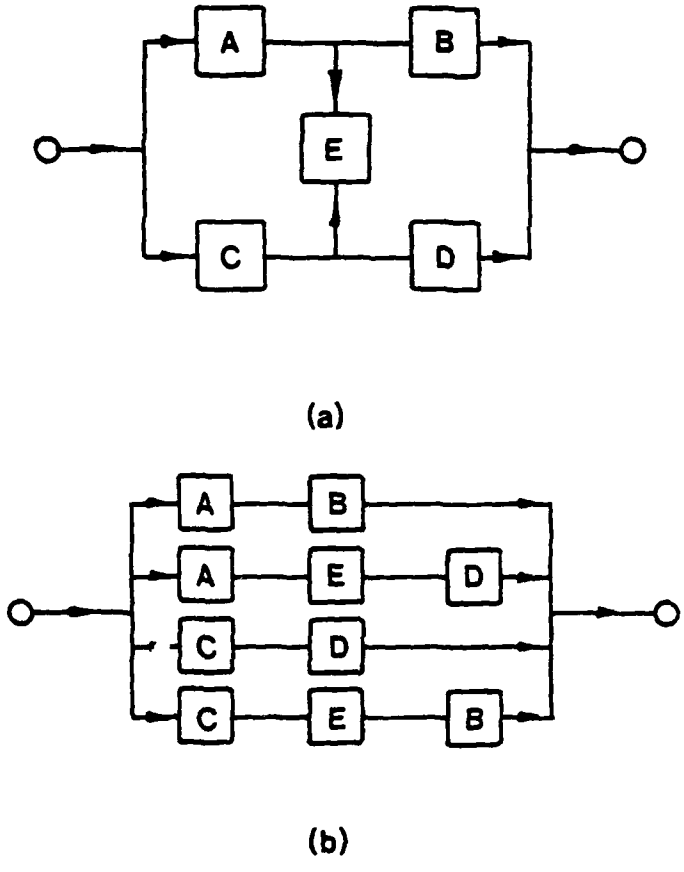


Figure 3-5: (a) A non series-parallel RBD, and
(b) its stochastically equivalent series-parallel RBD

required to construct one, namely connection of blocks, or groups of blocks, in series or in parallel. In addition any given block, rather than representing a single component in the system, may indeed represent an SPRBD for an entire subsystem of components. Hence it is easy to change the level of abstraction at which the system SPRBD is viewed by "hiding" entire sub-RBDs in single blocks in the main SPRBD.

- The simple series-parallel structure of an SPRBD suggests a simple method of obtaining the system reliability from it. Since blocks are connected either in series or in parallel one may consider "collapsing" single blocks in series (parallel) into one block representing an event which is the intersection (union) of the events represented by the two blocks individually. Thus, working from the inside out, an SPRBD may be collapsed into a single block whose probability will then represent the system reliability. Two simple rules were developed to use this collapsing process to produce a symbolic reliability function. They are described below. The

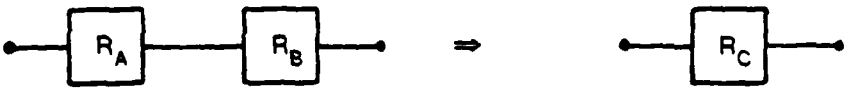
strong advantage of the algorithms using these rules is that they are very simple and robust. By robustness here we mean the ability to tolerate logically redundant event specifications and still compute the correct reliability function. Such stability is very useful in the context of automatic reliability function generation where the various subexpressions are generated by different parts of the program at different times during the computation. It allows the various knowledge based solvers in the reliability function generator to be independent of one another since the robustness of the intermediate representation algorithms guarantees that any redundancy in their generated subexpressions will be tolerated and accounted for.

3.3.1 The model underlying the SPRBD

In this section we put forward the assumptions under which the SPRBD algorithms described below will work. Components in the system are presumed to be binary state devices, i.e. either a component is functional or it is failed. If the system is composed of N components, therefore, there are 2^N possible system states. We may view the system state as an N-dimensional binary vector. If the set of N system components is supposed as ordered then each component is assigned a unique bit position in the binary vector. If component i is functional in some state of the system then bit-position i of the binary vector for that state will contain a 1. Likewise, if component i is not functional in some system state then the i^{th} bit-position of the state vector will contain a 0. The sample space, on which our probabilities are defined, is the set of 2^N possible system states. The simplest events in this sample space which are of interest to us are component successes. We shall term these Primitive Events. The primitive event that component i is successful is composed of the set of system states (i.e. outcomes in the sample space) for which bit-position i in the state vector contains a 1. By our basic assumption in Chapter 2 components have statistically independent failure behavior. Therefore, it is easy to see that our primitive events are all statistically independent. We shall term as Complex Events those events which are composed of some function of unions and/or intersections of primitive events. Clearly, two complex events can be stochastically dependent if the subsets of primitive events which compose them overlap. We shall assume that all events, whether primitive or complex, are assigned unique symbolic labels such as "Ev_x" from the countable set . The probability of occurrence of Ev_x will then be the unique symbol or label "R_x" from the countable set . It is evident that any complex event Ev_x is expressible as a regular expression over . The symbolic probability R_x will be expressible as a polynomial over the symbolic probabilities of the primitive events constituting Ev_x. If the complex event we are considering is system success then that polynomial is the symbolic system reliability function. With these preliminaries we shall proceed to a description of the SPRBD algorithms.

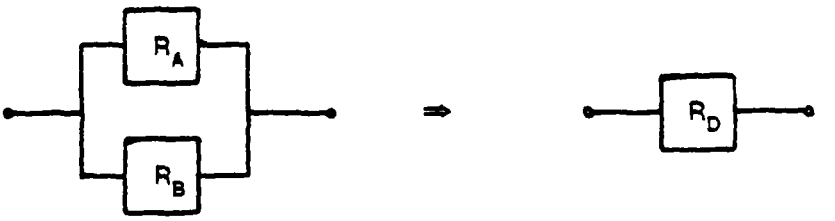
3.3.2 The SPRBD Algorithms

In an SPRBD there are only two types of connections possible for blocks and these are shown in Figure 3-6(a) and (b). Each of them may be "collapsed" or "merged" when encountered to produce a single block with the appropriate symbolic labels computed as shown in the figures. We shall explain the symbolic computations and in particular the "⊗" operator in the following.



$$R_C = R_A \otimes R_B$$

(a)



$$R_D = R_A + R_B - R_A \otimes R_B$$

(b)

Figure 3-6: Merging rules for SPRBDs

In Figure 3-6 the blocks connected in series or parallel may each represent either a

primitive event or a complex event. In the former case the symbol R_A, say, would just be the symbolic probability of the primitive event. In the latter case the symbol R_A would name a symbolic expression which was the symbolic probability of the complex event. From the foregoing discussion of Reliability Graphs it follows that a series connection of two blocks implies the intersection of the corresponding events while a parallel connection implies the union of them. The symbolic computations given in Figures 3-6(a) and (b) therefore compute the symbolic probability of the intersection and union of two events respectively. They are repeated here for convenience:

$$R_C = R_A \otimes R_B \qquad \text{(SMERGE} \Rightarrow \text{Serial Merge)}$$

$$R_D = R_A + R_B - R_A \otimes R_B \qquad \text{(PMERGE} \Rightarrow \text{Parallel Merge)}$$

At least superficially these rules are very similar to basic expressions in probability theory for computing the probabilities of intersections and unions of events. The $\otimes$ operator is intended to compute the symbolic probability of the intersection of two events given their individual symbolic probabilities. For this reason we shall term the $\otimes$ operator the Symbolic Intersection Probability or SIP operator. The "+" and "-" operators in Figure 3-6 have their usual algebraic meanings of addition and subtraction.

The symbolic expressions which represent the probability of the complex events will belong to a restricted class of polynomials which we shall term Canonical Reliability Polynomials (CRPs). We define them recursively thus:

Definition 3.1:

1. Individual atomic symbols such as R_A, R_B, R_C,... are Canonical Reliability Polynomials.

2. If R_A and R_B are Canonical Reliability Polynomials then so are

$$f_S(R_A,R_B) \equiv R_A \otimes R_B, \text{ and} \qquad \text{(SMERGE)}$$

$$f_P(R_A,R_B) \equiv R_A + R_B - R_A \otimes R_B \qquad \text{(PMERGE)}$$

3. A formula is a Canonical Reliability Polynomial iff it is formed in accordance with 1 and 2.

We now define the SIP operation as follows:

Definition 3.2: The $\otimes$ (SIP) operator

- Case 1: Ev_A and Ev_B are primitive events and their unique probability symbols R_A and R_B are atomic. The probability of the event $(Ev_A \cap Ev_B)$ is then simply written $R_A \otimes R_B$ and

$$R_A \otimes R_B = R_B \otimes R_A \qquad \text{(Commutativity)}$$

If the events Ev_A and Ev_B are primitive then $R_A \otimes R_B \equiv R_A \times R_B$.

- Case 2: Ev_A, Ev_B and Ev_C are primitive events and their unique probability symbols are R_A, R_B and R_C respectively, then

$$(R_A \otimes R_B) \otimes R_C = R_A \otimes (R_B \otimes R_C) = R_A \otimes R_B \otimes R_C \qquad \text{(Associativity)}$$

- Case 3: The $\otimes$ operator is idempotent, i.e.

$$R_A \otimes R_A = R_A \qquad \text{(Idempotency)}$$

Ev_A and Ev_B are complex events which are composed purely of the intersection of primitive events

$$Ev_A = \cap_{j=1}^{r} Ev_{aj} \quad (Ev_{aj} \text{ primitive})$$

$$Ev_B = \cap_{k=1}^{s} Ev_{bk} \quad (Ev_{bk} \text{ primitive})$$

Then their CRPs consist of one term each and are given by

$$R_A = C_a R_{a1} \otimes R_{a2} \otimes .... \otimes R_{aj} \otimes .... \otimes R_{ar}$$

$$R_B = C_b R_{b1} \otimes R_{b2} \otimes .... \otimes R_{bk} \otimes .... \otimes R_{bs}$$

where $C_a = +1$ and $C_b = +1$ are integer coefficients and $r>0$ and $s>0$. The probability of the event $Ev_C = (Ev_A \cap Ev_B)$ is given by

$$R_C = C_c R_{c1} \otimes R_{c2} \otimes ... \otimes R_{ci} \otimes .... \otimes R_{ct}$$

where $C_c = C_a C_b$, $t \leq r + s$, $t>0$ and $\{R_{ci}\} = \{R_{aj}\} \cup \{R_{bk}\}$

Thus, for example[10]

$$R_A = R_1 \otimes R_2 \otimes R_3 \otimes R_6$$

$$R_B = R_3 \otimes R_4 \otimes R_5 \otimes R_6$$

$$R_C = R_A \otimes R_B = R_1 \otimes R_2 \otimes R_3 \otimes R_4 \otimes R_5 \otimes R_6$$

- Case 4: Ev_A and Ev_B are complex events composed of unions and/or intersections of other simple or complex events. Each of them will have CRPs with more than one term. In this case the CRP resulting from the

---

[10] The effect of idempotency is similar to that of the operation defined in [Kim 72]. The difference is that in ADVISER the CRPs (which are analogues of reliability block diagrams) are constructed by the program instead of manually.

operation $R_A \otimes R_B$ will consist of a set of terms which is the Cartesian product of the individual sets of terms in R_A and R_B. Thus if R_A has m terms and R_B has n terms, $R_A \otimes R_B$ will have mxn terms each of which will result from an application of Case 3 above. If two or more of these mxn terms differ only in their integer coefficients then they may be replaced by one term with the same factors and an integer coefficient which is the algebraic sum of the coefficients of the replaced terms. (Distributivity)

It is evident from the definition of Canonical Reliability Polynomials and the SIP operator that a representative term of such a polynomial is of the form

$$C_a R_{a1} \otimes R_{a2} \otimes R_{a3} \otimes .... \otimes R_{aj} \otimes .... \otimes R_{ar}, \quad r>0 \tag{3.1}$$

where $C_a \neq 0$ is the integer coefficient of the term and R_{aj} are the probabilities of occurrence of the events aj which may or may not be complex events. If some event aj is complex then R_{aj} names a CRP which is the symbolic probability of aj in terms of the symbolic probabilities of other primitive (or complex) events. In this case the CRP named by R_{aj} in the expression (3.1) must be substituted into the term in place of the symbol R_{aj} and the indicated $\otimes$ operation carried out.

It is also to be noted that if all the R_{aj} in expression (3.1) represent the probabilities of primitive events *or complex events which are independent* then the $\otimes$ may be replaced by simple multiplication. Note that two complex events in our scheme will be independent if their sets of constituent primitive events have a null intersection. If two CRPs are in their simplest form, i.e. all factors in all terms of the CRPs represent the probabilities of primitive events, then independence of the corresponding two events may be deduced if the two CRPs have no factors in common in any of their terms.

3.4 A data structure for the SPRBD algorithm

We describe in this section the data structure which was chosen to represent Canonical Reliability Polynomials. Since the reliability expressions are in canonical form and due to the idempotency of the SIP operator, none of the unique literal symbols in a CRP will be raised to greater than unity power. In addition, literal symbols are either in a polynomial term or they are absent (for instance they do not appear in complemented form). This naturally suggests that a bit vector may be used to represent a term in a CRP. A unique bit position in the vector would be assigned to each unique symbol. Then a factor is present in a term if its bit is set to 1 and not present if its bit is set to 0. Furthermore, each term in the polynomial has a signed coefficient and one extra bit would be taken to represent this sign. Finally, the number of

primitive events is known to be the number of components in the system and the length of the bit vector for representing primitive events is thus also known. After a certain point in the calculation being performed by ADVISER it is also known how many complex event CRPs are to be manipulated and thus unique bits may be assigned for them at that point also (see Chapter 6).

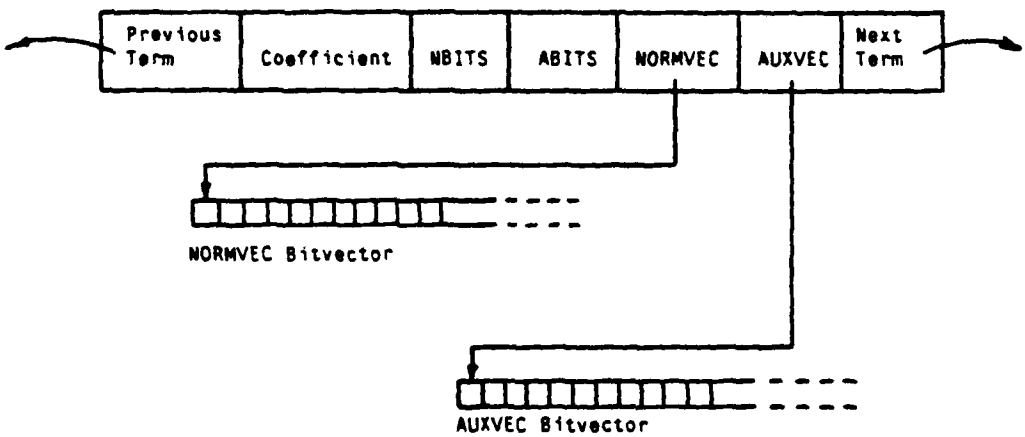


Figure 3-7: Data structure for CRP term

Figure 3-7 shows the logical data structure that results for a single Canonical Reliability Polynomial term. Since addition and subtraction are commutative the CRPs may be thought of as unordered sets of terms each with a signed coefficient.[11] Each such set of terms is represented in ADVISER by a doubly-linked list in which each element is of the form shown in Figure 3-7.

We now describe the fields in the data structure shown in Figure 3-7.

- The NORMVEC field of the term points to a bit vector which has as many bits as there are system components. As discussed above, each of these bits corresponds to one primitive event as we have defined it. Either the NORMVEC bit vector or the AUXVEC bit vector (see below) or both must be present in any term.

[11] possibly multisets if terms with identical factors have not been replaced by a single like term whose coefficient is the sum of their coefficients.

- Likewise, the AUXVEC field points to a bit vector which has as many bits as there are CRPs which were generated as intermediate results in the computation. The bits in the AUXVEC bit vector when set to 1 indicate that the corresponding CRP must be back substituted in the final system CRP in order to get the system reliability function. This process of back substitution must take into account any intersection between the complex event represented by the CRP being substituted and events represented by the other factors in the term (see Chapter 6). Either the AUXVEC bitvector or the NORMVEC bit vector (see above) or both must be present in any CRP term.

- The NBITS and ABITS fields are used for efficiency and hold the count of 1-bits (i.e. the number of factors) in the NORMVEC and AUXVEC bit vectors respectively. The need for these fields is described below.

- The MCONST field holds the signed integer coefficient of the of the term.

- The NEXTERM and PREVTERM fields point respectively to the next and previous terms in the list of such terms which comprise a CRP.

3.4.1 Ordering of CRP terms

The SMERGE and PMERGE rules of Figure 3-6 involve generating the Cartesian product set of the sets of terms of the two CRPs being merged. Thus the complexity of the $\otimes$ operation on two CRPs is $O(n^2)$ where n is the number of terms in a CRP. Furthermore, if the lists are unordered, the process of finding terms of like factors to add coefficients is $O(n^2/2)$. However, this latter cost is reduced if the lists of terms are kept ordered using some precedence function which compares terms based on the factors they contain. Then terms which will cancel or add will occupy adjacent positions in a list and these operations will cost less.

The particular precedence function which is chosen for the ordering must of necessity be simple to compute so as to minimize the time taken for doing ordered insertion of terms into lists, etc. In this particular case the NBITS and the ABITS fields were used to compare terms. For any two CRP terms a and b, a was taken to *precede* b if it had *fewer* factors than b. If a and b have an equal number of factors then the precedence was left undefined. This imposes a partial order on the CRP term lists such that terms in a CRP are arranged in order of increasing numbers of factors. Terms of equal numbers of factors will be found adjacent to one another but in order to distinguish among them a strict equality test must be performed on their sets of factors. In the implementation the NBITS and ABITS fields were placed adjacent to one another in the data structure thus allowing the precedence function to be computed in at most two instructions. The equality test, however, depends on the length of the NORMVEC and AUXVEC bitvectors.

3.5 An implementation of the SMERGE algorithm

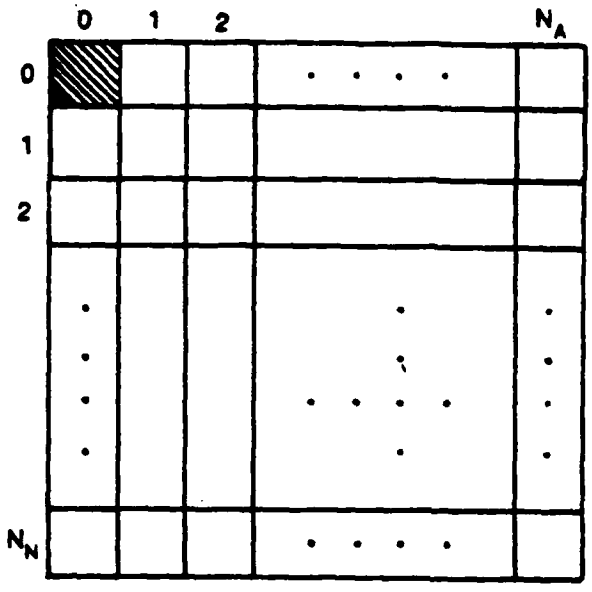
It may be observed from Figure 3-6 that the basic operation in the merging of both types of primitive SPRBD connections into a single block is the ⊗ operation. This is all that is required in the SMERGE operation shown below

$$R_C = R_A \otimes R_B \qquad \text{(SMERGE)}$$

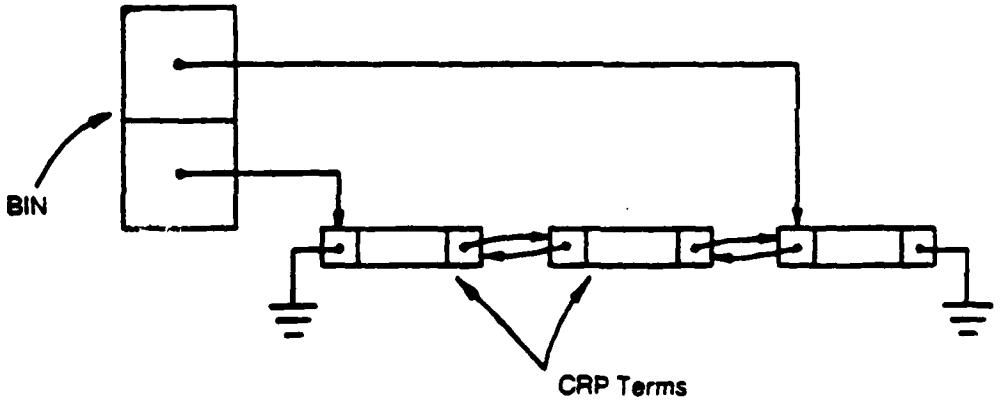
In this case the set of terms of the CRP R_C consists of the Cartesian product of the sets of terms in R_A and R_B under the ⊗ operation and after cancellation/adding of terms has taken place. This is implemented as described below.

The precedence of a CRP term may be characterized by the values of its NBITS and ABITS fields. Let these values be b_N and b_A respectively. Assume that for a given system there are N_N components which is thus the length of the NORMVEC. Also assume that N_A intermediate result CRPs are generated by ADVISER. Then it is evident that $0 \leq b_N \leq N_N$ and $0 \leq b_A \leq N_A$ although it is not permissible to have $b_N = b_A = 0$. Therefore, any term of any CRP generated during the reliability computation for the given system may be cast, on the basis of its b_N and b_A values as row and column indices , into one of an array of $(N_N + 1)x(N_A + 1)$ bins as in Figure 3-8(a). No term will fall into bin [0,0] and it is thus cross-hatched in the figure. Terms which fall into the same bin all have the same number of factors and are not in any specified order. When a term is inserted into a bin it is compared for strict equality of factor sets with each term on the list in the bin. The process stops when either the end of the list has been reached, in which case the new term is appended to the list, or the equality test succeeds. If the latter, then the term in the list which matched is replaced by the algebraic addition of the two terms. On the average the lists of terms in the bins are expected to be shorter than the length of CRPs otherwise and the time complexity will correspondingly be reduced.

The bin array is used as a device to implement the SMERGE algorithm so that the resultant CRP is an ordered list. This is done as follows. When the Cartesian product set of terms is being formed, each resultant term is cast into its respective bin where an insertion is performed as described above to put it in its correct place in the list of terms in the bin. A representative bin is shown in Figure 3-8(b). The algebraic addition of terms of equal factor sets in a bin also serves to hold down the number of terms in lists in bins and in the final output list. At the end of the cross-product process the bins are emptied out in order of precedence which is determined by their row and column indices. Thus Row 0 is of highest precedence followed by Row 1 etc. Bin [0,0] does not participate. Within each row the



(a)



(b)

Figure 3-8: (a) The Bin Array (b) A representative bin.

precedence decreases from left to right (see Section 3.4.1). As each bin is emptied, the ordered list held within it is concatenated onto the end of the output list of the SMERGE algorithm. Thus when the process concludes, the output list of CRP terms will be in precedence order. The SMERGE algorithm is shown below.

Algorithm SMERGE

Terminology:

- N_N, the number of components in the system; also the number of rows in the Bin Array.
- N_A, the number of intermediate result CRPs generated by ADVISER; also the number of columns in the Bin Array.
- R_A, R_B, R_C. The algorithm computes $R_C = R_A \otimes R_B$.
- t_k is the k^{th} term of a CRP; c_k is the integer coefficient of t_k, f_k is the set of factors of t_k, and b_{Nk} and b_{Ak} denote the NBITS and SBITS field values of t_k.
- α, β, γ are the sets of terms in the CRPs R_A, R_B and R_C respectively.
- $\omega[i,j]$ are the contents (set of terms) of the $[i,j]^{th}$ bin in the Bin Array.
- $+_{ins}$ is a binary operation and denotes insertion of a CRP term (right operand) into a list (left operand) as described in the text above.
- $+_{conc}$ is a binary operation and denotes concatenation of a list (right operand) to the end of another list (left operand).
- $\emptyset$ denotes the empty set.

```
Procedure SMERGE
Begin

Incr i from 0 to N_N do
    Incr j from 0 to N_A do ω_i,j ← Ø od od;
            Comment make the bins empty.

Foreach i suchthat t_i ∈ α do
    Foreach j suchthat t_j ∈ β do
        c_k ← c_i x c_j;
        f_k ← f_i ∪ f_j;
        ω[b_Nk, b_Ak] ← ω[b_Nk, b_Ak] +_ins t_k
    od
od;     Comment form Cartesian product set using ⊗
                and insert terms into their bins;

γ ← Ø;
      Comment clear the output list;

Comment empty out each bin in precedence order and append its
    contents to the output list, ignore ω[0,0];

Incr j from 1 to N_A do
    γ ← γ +_conc ω[0,j] od;

Incr i from 0 to N_N do
    Incr j from 0 to N_A do
          γ ← γ +_conc ω[i,j]
    od
od;


End; Comment end of algorithm SMERGE;
```

3.6 An implementation of the PMERGE algorithm

The ⊗ operator is also fundamental to the PMERGE rule of Figure 3-6. The PMERGE rule is shown below:

$$R_D = R_A + R_B - R_A \otimes R_B \qquad \text{(PMERGE)}$$

It is evident that in order to PMERGE R_A and R_B we must first compute the SMERGE of the two CRPs. This is done exactly as described in the previous section. The only difference here is that each of the terms resulting from the $R_A \otimes R_B$ operation must have its coefficient negated. Subsequently, the sets of terms of the CRPs R_A, R_B and $R_A \otimes R_B$ are all pooled to form the set of terms for the result R_D. This must be done so that the resulting list is also ordered according to precedence order. For this purpose a simple three-way list merging technique is used ([Knuth 75a]). Each of the three lists may be viewed as linked stacks. The top elements of the three stacks are compared for precedence and the term with the highest precedence is "popped" off its stack and concatenated to the end of the output list. If two, or all three, terms at the tops of the stacks have identical factor sets then they are popped off their respective stacks and algebraically added. The resulting single term is then concatenated to the end of the output list.

3.7 Summary

This chapter introduced the Canonical Reliability Polynomial (CRP) as the basic representation in ADVISER for the symbolic probabilities of occurrences of events in the model. A list representation for CRPs was described, as were two simple algorithms to manipulate this representation. These algorithms, named SMERGE and PMERGE, respectively compute the symbolic probabilities of the intersections and unions of events in the model, given their individual probabilities expressed in CRP form. The algorithms are robust in that they are tolerant of overspecification. Thus ADVISER need not keep track of the history of construction of any two CRPs which are merged using these algorithms. Even if both CRPs state the probability of the same event, the idempotency of the SIP operator, ⊗, ensures that the correct intersection, or union, probability will be computed. The algorithms serve as straightforward tools for use in ADVISER during the incremental construction of the symbolic system success probability from the reliability symbols of the individual system components, and the operational requirements. The simplicity and ease of use of these algorithms enabled the modular construction of ADVISER among other benefits. However, Chapter 7 shows that the efficiency of these intermediate representation algorithms, though

not dismal, could stand improvement. Although not suitable in their current state, algorithms of the type described in [Satyanarayana 78], [Aggarwal 78], [Bennetts 75] and [Lin 76], could possibly be candidates for replacing SMERGE and PMERGE. However, modification would be necessary and it would have to be shown that the efficiency of the replacement is superior to that of SMERGE and PMERGE.

Chapter 4
Detection of symmetries in the PMS graph

We will be concerned in this chapter with the ability to discover symmetric subsystems within computer systems. The discovery of such symmetric subparts allows economies in the reliability calculation process. Figure 4-1 shows the portion of ADVISER which is described in this chapter.

Current trends in the design of computer systems are toward multiple processor systems of various kinds. From the point of view of ease of design as well as modularity and ease of maintenance it is convenient to build such structures from symmetric subunits. For instance, in the PLURIBUS multiprocessor [Ornstein 75] there are three kinds of symmetric subsystems, namely, processor buses, memory buses, and input/output buses. There may be more than one of each kind in a PLURIBUS multiprocessor and they may be connected together in semi-arbitrary fashion. Another example is the *Cm* multiprocessor* [Swan 77] which is composed of processor-memory pairs connected into clusters which in turn may be connected in some arbitrary fashion. These types of structures, therefore, have interconnection graphs which contain symmetric subgraphs. One also finds symmetric substructures within uniprocessor systems which have replicated input/output subsystems for availability purposes. However, the reverse is not true. Symmetric subgraphs in the interconnection graph do not necessarily imply what we intuitively conceive physical symmetry to be. Two sets of *completely different types of components* may each be connected in an identical interconnection pattern. See Figure 4-4 for two PMS structures which, though isomorphic, do not satisfy our intuitive notions of symmetry for physical structures. For this reason, the graph model of a PMS structure is more appropriately viewed as a labelled graph. The label of each vertex in the graph associates it with a component of a particular physical type. We suppose two components to be identical, in their reliability behavior at any rate, if they are of the same type. We shall, for instance, classify a PDP-11/40[12] and a PDP-11/45 to be two different types of components since they presumably have different failure rates although they are both CPUs

[12] PDP is a registered trademark of Digital Equipment Corporation.

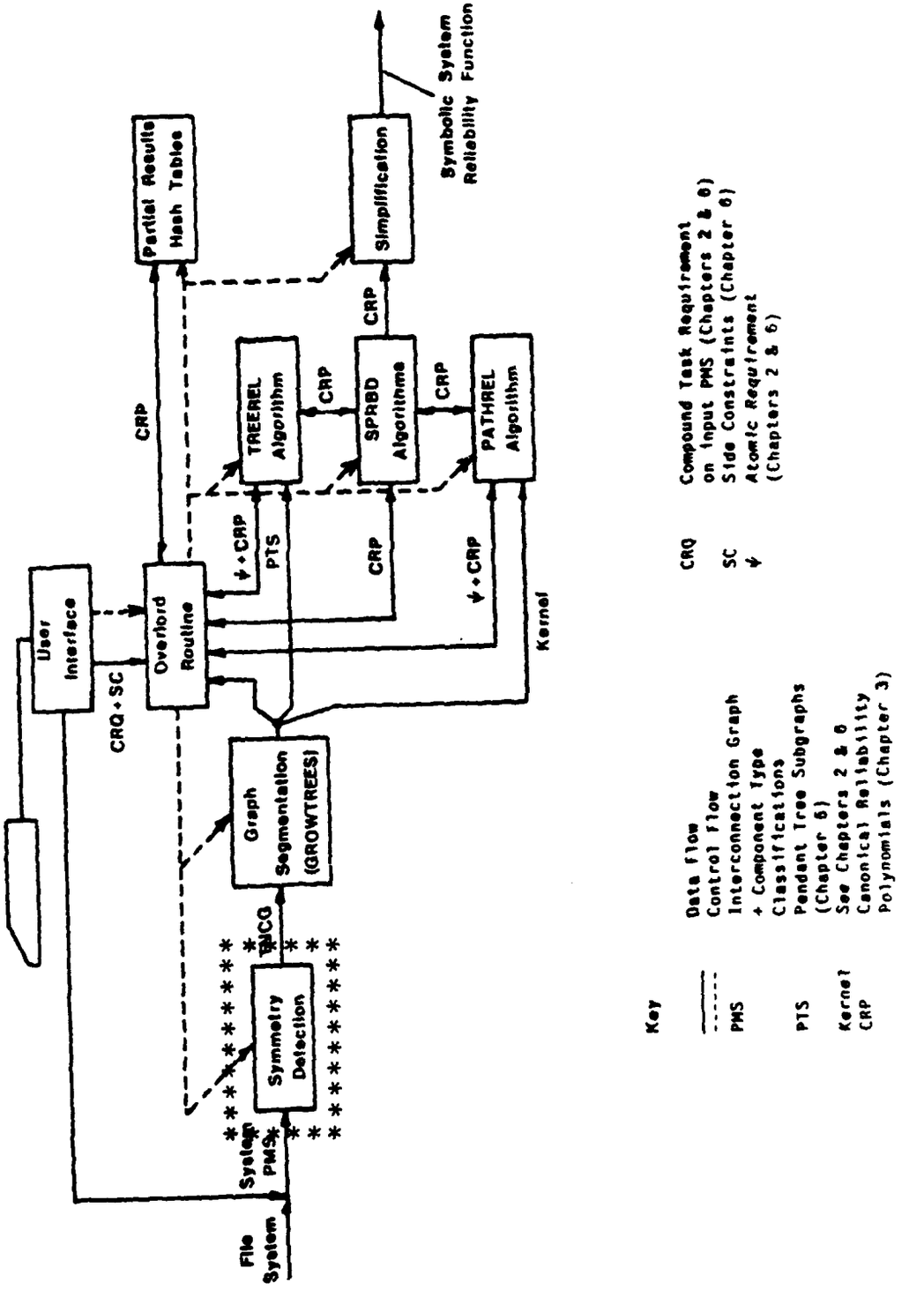


Figure 4-1: The portion of the ADVISER structure discussed in Chapter 4.
Also see Page 18.

and PDP-11s in a functional sense. However, two PDP-11/40s will be considered to be identical since they are of the same type. At any rate the goal is to have a basis whereby any two components in a PMS structure may be compared, to be subsequently found to be either identical or different in their reliability behavior. The component type mechanism and the labelling of each interconnection graph vertex with the type of its component provides this basis.

> **Definition 4.1: Physical Symmetry:** We shall consider two graphs to be "physically symmetric" iff they are isomorphic and the corresponding vertices of the two graphs have identical component type labels.

Consequently, if the process of finding symmetric subgraphs in the PMS interconnection graph takes vertex labels into account, the symmetries detected will correspond in unique fashion to the physical symmetries in the system.

We are thus led to consider algorithms for generating the symmetric subgraphs of labelled subgraphs. The next section introduces an algorithm for partitioning of the vertex set of an unlabelled graph into equivalence classes based on structural symmetries within the graph. In such graphs the vertices are homogeneous and any symmetries are thus based on connectivity only. Subsequent sections will modify this algorithm for the case of a labelled graph thereby introducing the labels of the vertices as an additional factor to determine symmetry. Finally some properties of the partition into equivalence classes will be described.

We shall assume henceforth that the graphs being considered are finite and have no multiple edges, ie. any two vertices which are immediate neighbors will not have more than one edge connecting them. The definitions and results presented here refer to non-directed graphs since these are the basis for our model. They may be extended to strongly connected directed graphs (see [Gaschnig 77]).

I am deeply indebted to John Gaschnig, now at SRI International, Menlo Park, CA., for an introduction to the ideas in this section. Results attributable to him are so marked. However, the responsibility for any errors or omissions is entirely mine.

4.1 A symmetry detection algorithm based on equivalence classes

In this section we shall consider unlabelled graphs i.e. those whose vertices are homogeneous. Intuitively, the search for structural symmetries in graphs must begin with the notion that two corresponding vertices of two symmetric subgraphs must have at least the

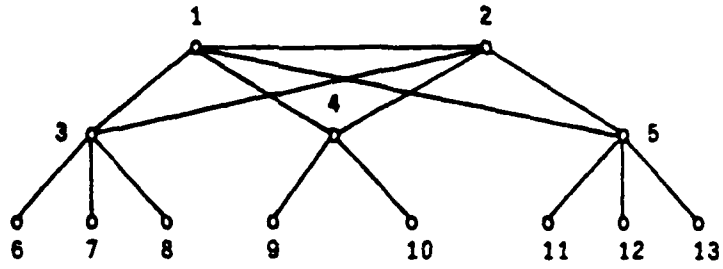
same degree. It is then possible to begin by partitioning the set of vertices of a PMS interconnection graph G(V,E) into equivalence classes based on this observation. We shall subsequently introduce the Neighbor Class Equivalence Relation of Gaschnig and finally modify it for the kinds of graphs we intend to study ie. labelled, non-directed graphs. Henceforth, let the notation "$x \equiv_R y$" mean that x is equivalent to y under the equivalence relation R, ie. x and y would fall into the same equivalence class in a partition induced by R. Likewise, let "$x \not\equiv_R y$" mean that x is not equivalent to y under R.

> **Definition 4.2:** (Gaschnig) Equal Degree Equivalence Relation (ED). Let G(V,E) be a non-directed graph and let R be an equivalence relation on V. R is said to be an equal degree equivalence relation iff $\forall u,v \in V$, $u \equiv_{ED} v$ iff $d(u) = d(v)$, where $d(x)$ is the degree of vertex x.

Regular graphs, wherein all vertices have the same degree, will have their vertices fall into exactly one class by virtue of the ED relation. This is still not satisfactory from the point of view of finding symmetries since, in general, it is possible for two vertices to be of equal degree and still be connected to subgraphs which are not isomorphic. Thus we need to equivalence two vertices if they are of equal degree and, in addition, the respective subgraphs to which they are connected are isomorphic. This may be achieved by introducing an equivalence relation on the vertices of G which has the property that two vertices are equivalent iff they are of equal degree and the number of their neighboring vertices belonging to each equivalence class due to the relation is the same.

> **Definition 4.3:** (Gaschnig) Neighbors Class Equivalence Relation (NCER). Let G(V,E) be a non-directed graph and let R be an equivalence relation on V. Arbitrarily name the equivalence classes of V due to R by the distinct symbols $c_1, c_2, \ldots, c_m$. Let c(v) denote the name of the equivalence class in which vertex $v \in V$ belongs. Define the neighbors class of a vertex v to be the set $NCM(v) \equiv \{c(w) | (v,w) \in E\}$. Then, R is a Neighbors Class equivalence relation (or R is NCER) under the following condition:
>
> $$\forall\, u,v \in V,\ v \equiv_R u \text{ iff } NCM(v) = NCM(u)$$

Several elementary properties of this relation are immediately apparent. For any graph the partition wherein each vertex falls into its own equivalence class is trivially NCER. Consequently, if symmetries exist, in general, it is possible for a graph to have more than one partition which is NCER. For regular graphs the partition consisting of a single equivalence class is NCER. Equal degree is a necessary condition for NCER equivalence of vertices so that $d(u) \neq d(v) \Rightarrow u \not\equiv_{NCER} v$. We shall use a simple example to explain the effect of the NCER and as an introduction to an algorithm to generate an NCER partition of a graph. Consider the graph shown in Figure 4-2. It is evident to the eye that symmetries exist within it.



Equal Degree Partition

Step 1:

$C_1 = \{3\ 5\}_{d=5}$, $C_2 = \{1\ 2\ 4\}_{d=4}$, $C_3 = \{6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\}_{d=1}$

Iterative Splitting to achieve NCER partition

Step 2:

$C_1 = \{3\ 5\}$, $C_2' = \{1\ 2\}$, $C_2'' = \{4\}$, $C_3 = \{6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\}$

Step 3:

$C_1 = a = \{3\ 5\}$, $C_2' = b = \{1\ 2\}$, $C_2'' = c = \{4\}$,

$C_3' = d = \{6\ 7\ 8\ 11\ 12\ 13\}$, $C_3'' = e = \{9\ 10\}$

Neighbors Class Adjacency Matrix (NCAM)

Step 4:

	a	b	c	d	e
a	0	2	0	3	0
b	2	1	1	0	0
c	0	2	0	0	2
d	1	0	0	0	0
e	0	0	1	0	0

Neighbors Class Graph (NCG)

Step 5:

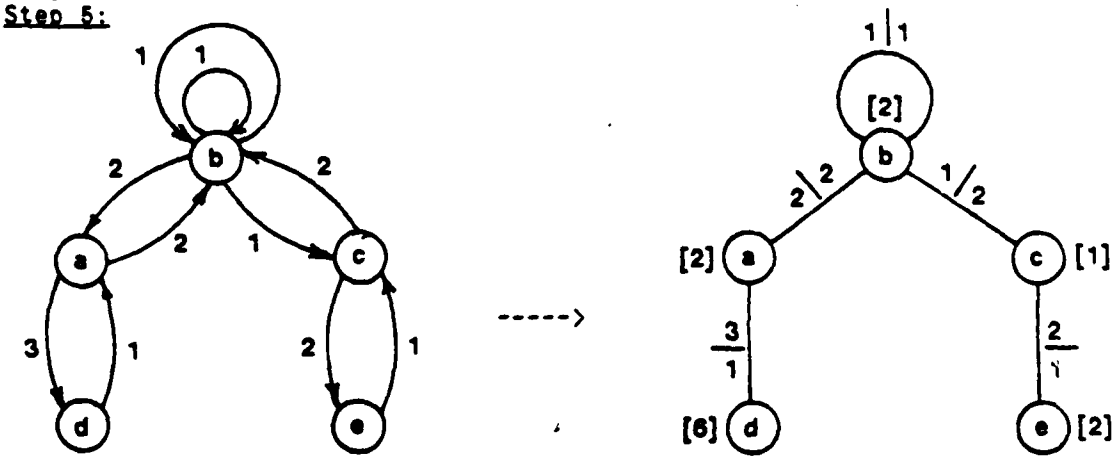


Figure 4-2: Application of the NCER to an example graph.

The detection of these symmetries proceeds as follows. The vertices of the graph are first partitioned into classes based on the equal degree (ED) relation. We shall term this the equal degree or ED partition. The ED partition for our example is shown at Step 1 in the figure. At this stage the NCMs of the various vertices are as follows:

$NCM(1) = NCM(2) = \{C_1\ C_2\}$

$NCM(3) = NCM(4) = NCM(5) = \{C_2\ C_3\}$

$NCM(9) = NCM(10) = \{C_2\}$

$NCM(j) = \{C_1\}\ \ j \in \{6,7,8,11,12,13\}$

It will be noticed, however, that this partition is not NCER since, for instance, $NCM(4) \neq NCM(1) = NCM(2)$ although vertices 1, 2 and 4 are in the same ED partition. This demonstrates that equal degree is only a necessary condition for NCER equivalence of two vertices. We now come to the notion of NC-consistency.

Definition 4.4: (Gaschnig) A class c_i of a partition $P = \{c_1,c_2,.....,c_m\}$ is said to be NC-consistent iff $\forall\ u,v \in c_i$, $NCM(u) = NCM(v)$. A class c_i is said to be NC-inconsistent iff it is not NC-consistent.

By these definitions, classes C_2 and C_3 in Figure 4-2 are NC-inconsistent whereas class C_1 is NC-consistent.

Having generated the ED partition, the algorithm proceeds by iteratively splitting each NC-inconsistent class into NC-consistent classes and then checking to see if any new NC-inconsistencies have been introduced in previously NC-consistent classes due to this splitting. The process continues until there are no more NC-inconsistent classes remaining. The algorithm then terminates and the resulting set of classes form an NCER partition of the graph. Referring to our example again, in Step 2 the NC-inconsistent class C_2 has been split into two NC-consistent classes C_2' and C_2''. However, this makes class C_3 NC-inconsistent (Note that class C_3 is already inconsistent to begin with in our example, however, even had it been consistent, this splitting of class C_2 would have made it inconsistent). In Step 3 the class C_3 has been split into two NC-consistent classes C_3' and C_3''. At this point all classes are NC-consistent and the algorithm terminates with the NCER partition $P_{NCER} = \{C_1,C_2',C_2'',C_3',C_3''\}$. We may characterize a class c_i in P_{NCER} in terms of the number of arcs from each vertex in c_i to its neighboring class. As for instance in the case of class C_2' wherein each of the vertices 1 and 2 have two arcs proceeding to their neighbor vertices in class C_1, one arc to class C_2'', and one arc to class C_2' (vertices 1 and 2 have an arc joining them). For expository purposes let us rename the classes $\{C_1,C_2',C_2'',C_3',C_3''\}$ as {a,b,c,d,e} respectively. We may then

construct a matrix as in Step 4 of Figure 4-2 which shows these connectivity relations. This is termed the Neighbors Class Adjacency Matrix (NCAM).

> **Definition 4.5:** (Gaschnig) The Neighbors Class Adjacency Matrix (NCAM) of an NCER partition $P = \{c_1, c_2, \ldots, c_m\}$ is a square matrix of size m with one row and one column corresponding to each class $c_i \in P$. $NCAM_{i,j} = k$ if exactly k vertices of class c_j are connected to *each* vertex of class c_i. *Note: for reasons of symmetry each vertex of class i will be connected to an identical number, $NCAM_{i,j}$, of neighbors in class j and the sets of neighbors in class j, of vertices in class i, may overlap.*

The definition of the NCAM very naturally leads us to the notion of a directed graph with weighted edges where the class names c_i, $i = 1,2,\ldots,m$ are its vertices and the NCAM is its adjacency matrix. Furthermore, $NCAM_{i,j} > 0$ is the weight of the edge joining the vertices representing the classes c_i and c_j respectively. This is termed the Neighbors Class Graph.

> **Definition 4.6:** (Gaschnig) Let $P = \{c_1, c_2, \ldots, c_m\}$ be an NCER partition of a graph G(V,E). Then the Neighbors Class Graph (NCG) of G is the graph $G'(V', E')$ where $V' = \{c_1, c_2, \ldots, c_m\}$ and for all ordered pairs (u', v'), $u', v' \in V'$, $(u', v') \in E'$ iff $NCAM_{u',v'} > 0$. Furthermore, for all $(u', v') \in E$ the weight of the edge (u', v') is the element $NCAM_{u',v'}$.

The NCG is a directed graph in which self-loops are allowed on vertices since it is quite possible to have a class be one of its own neighbors. Let $G'(V', E')$ be the NCG of G(V,E) and let $e' = (s', d')$ be an ordered pair such that $(s', d') \in E'$. Recall that s' and d' are equivalence classes of V. Then each directed arc such as e' in G' represents one or more arcs in G from each vertex of G in s' to its neighbor vertices in d'. The weight of e' is the number of such edges of G from a vertex in s' to its neighbors in d'. The NCG may alternatively be viewed as having multiple edges between its own vertices, the multiplicity being given by the edge weights. The NCG for our example is shown in Figure 4-2 on the left hand side of Step 5. On the right hand side of Step 5 in Figure 4-2 is an alternative, more compact, representation of the NCG. In this representation, the 1|2 on the edge between vertex b and vertex c implies that there is one edge from each vertex of class b to its neighbor vertices in class c, i.e. each vertex of b has one neighbor in c (in this case the same vertex 4 is neighbor to both 1 and 2), and, likewise, there are two edges from each vertex of class c to its neighbors in class b. In other words, the two weighted directed arcs between each pair of vertices in the original NCG have been collapsed into one non-directed edge with a dual weight which has a component in each direction.

> **Definition 4.7:** The connection density ρ_{xy} of an NCER equivalence class X with respect to its neighbor NCER equivalence class Y is the number of vertices in Y that are neighbors of each vertex of X.

The integer in brackets labelling each vertex of the NCG on the right hand side of Step 5 in Figure 4-2 is simply the cardinality of the corresponding class in the NCER partition. We shall derive some relations between these labels presently.

Returning to our algorithm for generating an NCER partition for a graph, we shall refer to the partition formed by it as the "Equal Degree then Split" or EDS partition in view of its nature. The algorithm for obtaining the EDS partition of G(V,E) is shown below. In addition, Figure 4-3 shows the effect of applying Algorithm EDS to various unlabelled graphs.

Algorithm EDS, ("Equal Degree, then Split")

Terminology:

- The graph under consideration will be G(V,E)
- Let n be the number of distinct classes into which the vertex set V is split by the Equal Degree (ED) relation and let the class names be the integers 1,2,.....,n.
- NCM(x) will denote the neighbor class set of the vertex x.
- The function firstelement(x) will denote some arbitrarily chosen "first" element of the unordered vertex class x.
- At the end of the algorithm the number of classes resulting will be held in the variable "last". In other words if the contents of the variable "last" is m upon termination, there will be m classes named 1,2,.....,m.

```
Procedure EDS
begin
     integer last, newlast;    Comment to hold class names;
     boolean done;
```

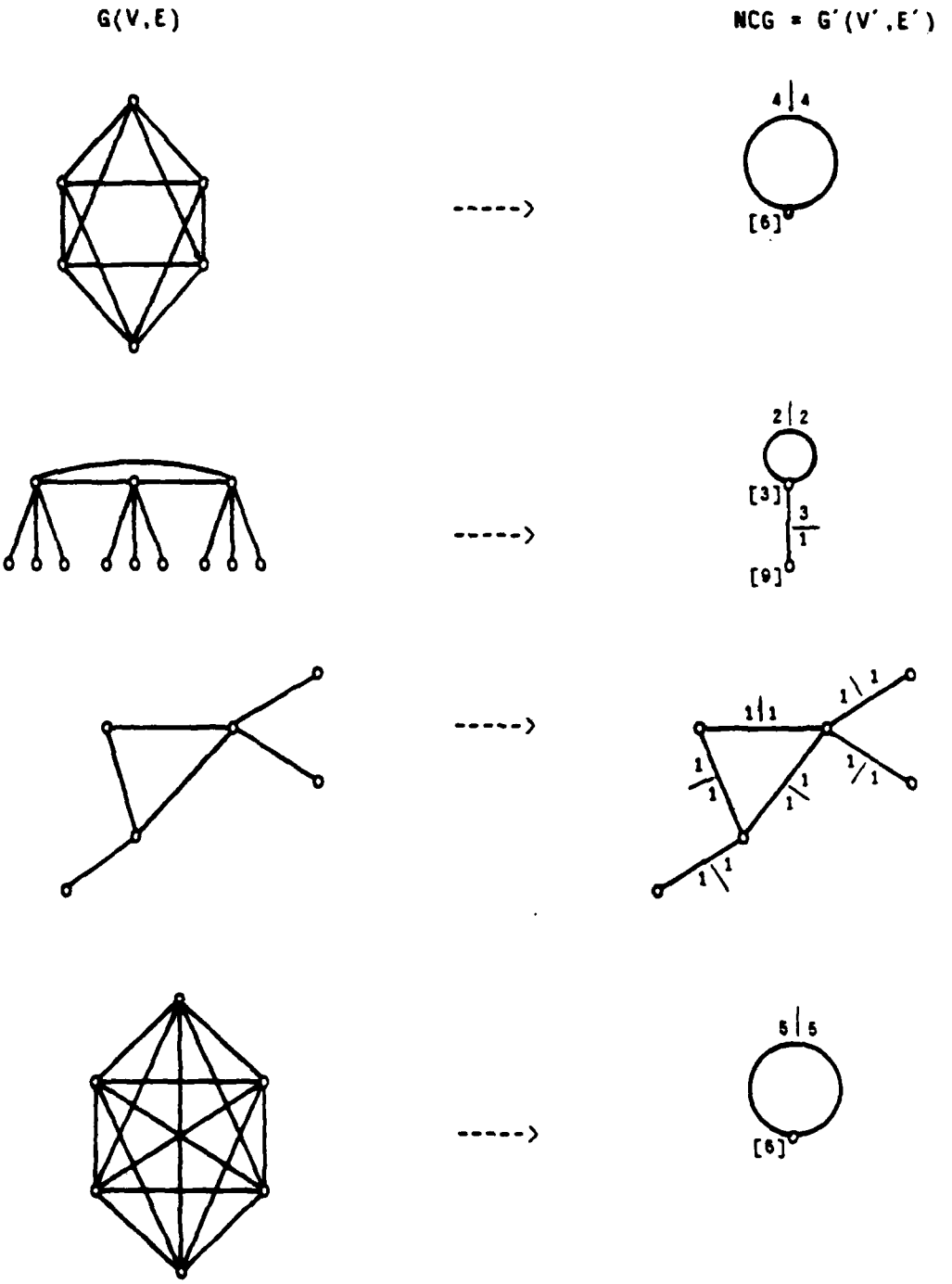


Figure 4-3: Examples of NCGs resulting from the application of NCER to various graphs.

```
Comment declare a procedure within EDS

Procedure SplitAClass (tc,nc)
begin
        boolean makenewclass;
        integer thisclass, newlastclass;
thisclass←tc;
newlastclass←nc;

if
    (cardinality of thisclass is unity)
then
    return thisclass
fi;


makenewclass←true;

while makenewclass
do
     if     (thisclass is NC-inconsistent)
     then
            newlastclass←newlastclass+1;
                        Comment create a new class;
            (initialize newlastclass to null set);
            foreach u∈V
            do
                  if NCM(u) ≠ NCM(firstelement(thisclass))
                  then (move u from the class thisclass
                              to the class newlastclass)   fi
            od;
            thisclass←newlastclass
     else makenewclass←false
     fi
od;
return newlastclass
end:                Comment end of Procedure SplitAClass
```

```
Comment the EDS algorithm begins here;

newlast←last←n;
done←false;

while not done
do
      for j from 1 to last
      do newlast←SplitAClass(j,newlast) od;

      if last = newlast        Comment no change;
      then done←true
      else last←newlast
      fi
od

end;    Comment end of Procedure EDS
```

4.2 Some properties of the NCER

This section presents some properties of the Neighbors Class Equivalence Relation. The proofs here are informal and are included for the purposes of exposition. For further details and a more rigorous treatment the reader is referred to [Gaschnig 77].

Theorem 4.1: [Gaschnig] In general an NCER partition of a non-directed graph G(V,E) is not unique.

Proof: For any non-directed graph G(V,E) the one partition P_0 wherein each $v \in V$ is assigned its own class is trivially NCER. If in this trivial NCER partition, two classes a and b exist such that NCM(a) = NCM(b) then a and b can be combined into one class. The resulting more compact partition P′ with one less class is still NCER by definition. Likewise, if there exists an NCER partition, P″ of G that is not trivial, consider a class c″ of P″ whose cardinality is greater than one. Then assigning each of the vertices of c″ to a class of its own, i.e. dividing c″ into as many classes as there are vertices in c″, will also generate an NCER partition. Thus an NCER partition for G is not unique. ∎

Theorem 4.2: [Gaschnig] The EDS partition is the minimal NCER partition.

Proof: The EDS algorithm will terminate as soon as all the classes produced thus far are NC-consistent. At each iteration of splitting and checking, only as many new classes are created from an NC-inconsistent class as are needed to satisfy NC-consistency of the old class and the new classes created from it. This happens in all but the final iteration before termination of the algorithm. ∎

Corollary 4.1: [Gaschnig] The EDS algorithm applied to the graph G(V,E) terminates after at most N-1 iterations, where N = |V|.

Some more properties of the Neighbors Class Equivalence Relation, and the partition induced by the EDS algorithm, are stated below without proof.

Theorem 4.3: [Gaschnig] If $P = \{c_1, c_2, \ldots, c_m\}$ is an NCER partition of G(V,E) then $\forall u,v \in V$, $u \not\equiv_{EDS} v \Rightarrow u \not\equiv_P v$

Theorem 4.3 implies that for each NCER partition P of G, each class of P is a subset of some class of the EDS partition. Thus, the NCER partition with the fewest classes is the EDS partition (cf. Theorem 4.2).

Theorem 4.4: [Gaschnig] Different graphs are mapped into isomorphic NCGs by the EDS algorithm.

This leads to the fact that the EDS algorithm is an "information reducing" operation and it is not always possible to deduce the graph which is the origin of an NCG. However, graphs which share the same "image" by having the same NCG, share common aspects although they may be very different in other ways. For instance all graphs which map into the same NCG will have identical proportions of vertices in each class. That the same NCG is produced implies that the same number of equivalences classes were produced by the partitioning of those graphs. As a result the number of vertices in any graph which maps into a given NCG will be an integral multiple of the number of vertices in the smallest graph which maps into the same NCG.

As Gaschnig remarks, the behavior of the EDS algorithm is analogous to that of a standard algorithm for sequential circuit state minimization, in which equivalent states are identified and replaced by a single equivalence class [Hill 68], pp.201-213.

4.3 Modification of EDS for labelled graphs

In this section we modify the EDS algorithm described above for the case of labelled graphs. Then we show the operation of the modified algorithm on the example PMS graph of Chapter 2, Page 29. Thus far the graphs we have studied were not labelled. Hence, the vertices of these graphs were all homogeneous. Accordingly, detecting structural symmetries in the graphs amounted to detecting their isomorphic subgraphs. However, the consideration of physical interconnection structures brings a new aspect to the meaning of symmetry. When

we speak of two interconnection structures being physically symmetric we imply that in addition to their interconnection graphs being isomorphic, they have identical types of components in the corresponding places in the structure. Note, again, that in our study of system reliability calculation we shall term two components to be of the same type if they are identical components with the same reliability functions. Thus in Figure 4-4 we see that the two structures are isomorphic in their interconnection graphs although they are not physically symmetric.

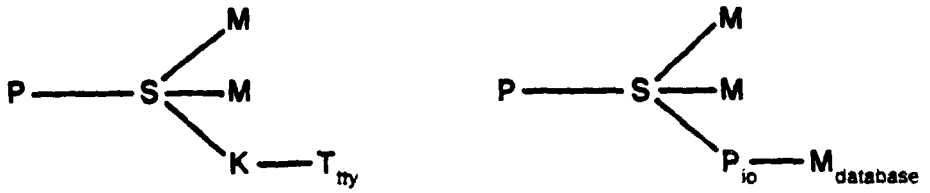


Figure 4-4: Non-symmetric but isomorphic PMS structures.

This additional constraint which determines physical symmetry may be incorporated very easily into the model by appropriately labelling each vertex of an interconnection graph by the type of the component it represents in the interconnection structure. We then have to modify Algorithm EDS for detecting symmetry in labelled graphs. The modification is simple and is expressed by the following slightly enhanced version of the NCER definition.

Definition 4.8: Typed Neighbors Class Equivalence Relation (TNCER). Let G(V,E) be a non-directed graph with labelled vertices and let R be an equivalence relation on V. Arbitrarily name the equivalence classes of V due to R by the distinct symbols $c_1, c_2, \ldots, c_m$. Let c(v) be the name of the equivalence class to which vertex $v \in V$ belongs. Let type(v) represent the label affixed to vertex $v \in V$. Define the neighbors class of a vertex to be the set $NCM(v) \equiv \{c(w) | (v,w) \in E\}$. Then R is a Typed Neighbors Class Equivalence Relation (TNCER) under the following condition:

$$\forall u,v \in V \; v \equiv_R u \text{ iff } NCM(u) = NCM(v) \wedge (type(u) = type(v))$$

We also define the following relation:

Definition 4.9: Equal Type Equivalence Relation. Let G(V,E) be a non-directed

graph and let R be an equivalence relation on V. R is said to be an Equal Type Equivalence Relation (ET) iff $\forall u,v \in V$, $u \equiv_{ET} v$ iff type(u) = type(v).

Then the modified EDS termed the ETEDS algorithm may be expressed as below. (Since it is so similar to Algorithm EDS only the major differences in the overall structure are shown.)

Algorithm ETEDS ("Equal Type, Equal Degree then Split")

```
Terminology: identical to Algorithm EDS

Procedure ETEDS
begin

integer last, newlast;
boolean done;

Procedure SplitAClass (tc,nc)
begin
...             Comment identical to Procedure SplitAClass in
...                         Algorithm EDS;
...
end;


Comment beginning of code for Procedure ETEDS

(Split V into equivalence classes based on Equal Type equivalence
  relation; let n classes c1,c2,....,cn result);

                Comment Step 1;


for i from 1 to n
do
      (Split ci into equivalence classes based on Equal Degree
    equivalence relation; let mi classes ci1,ci2,...cimi
    result)
od;           Comment Step 2;

Comment at this point a total of Σ(i=1 to n) mi classes have been generated.

newlast←last←Σ(i=1 to n) mi;
done←false;
```

```
while not done          Comment Step 3;
do
      for j from 1 to last
      do newlast ← SplitAClass(j,newlast) od;

      if last = newlast
      then done←true
      else last←newlast
      fi
od

end;       Comment end of Procedure ETEDS;
```

Theorem 4.5: The partition generated by Algorithm ETEDS is TNCER

Proof: After Step 1 in Algorithm ETEDS we have

$$\forall u,v \in c_i,\ i = 1,2,\ldots,n,\ \text{type}(u) = \text{type}(v)$$

After Step 2 in the algorithm we have

$$\forall u,v \in c_{ij},\ j = 1,2,\ldots,m_i,\ i = 1,2,\ldots,n,$$

$$(\text{type}(u) = \text{type}(v)) \wedge (d(u) = d(v))$$

The splitting process in Step 3 of the algorithm terminates only when all classes are NC-consistent. Thus for each class c at the end of Step 3

$$\forall u,v \in c\ (\text{type}(u) = \text{type}(v)) \wedge (d(u) = d(v)) \wedge (\text{NCM}(u) = \text{NCM}(v))$$

Therefore, the partition generated by algorithm ETEDS is TNCER. ∎

Analogous to the NCAM and the NCG in the case of the NCER we may define a TNCAM and TNCG in the case of the TNCER. In other words since two vertices fall into the same class of a TNCER partition iff their types are the same, we may label that *class* with the same type.

We now show the result of applying the ETEDS algorithm to the example PMS graph of Chapter 2, Figure 2-5. The graph is reproduced for convenience in Figure 4-5 and the symmetry detection steps applied to it are shown in Figure 4-6. In Step 1 of Figure 4-6 the vertex set of the graph has been partitioned according to the Equal Type (ET) equivalence relation. Step 2 shows the further partitioning according to vertex degree. Step 3 shows the iterative splitting, of partitions achieved so far, to obtain the equivalence classes due to the Typed Neighbors Class Equivalence Relation (TNCER). At each iteration an asterisk is used to

mark those classes which are NC-inconsistent. Figure 4-7 shows the resulting TNCAM and TNCG for the PMS graph of Figure 4-5.

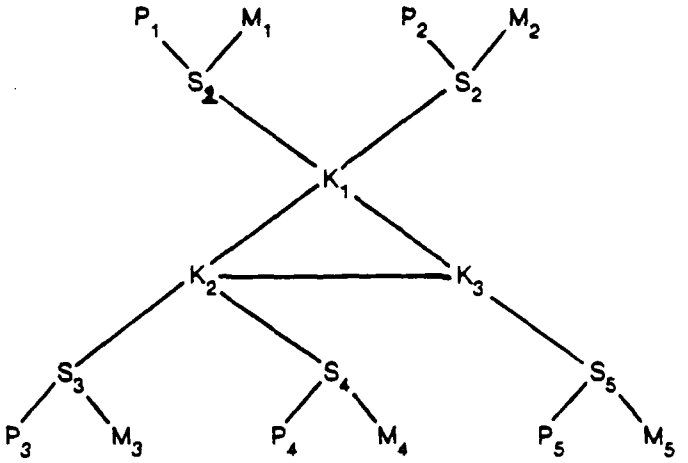


Figure 4-5: An example PMS graph with symmetries

4.4 Some results in regard to the NCG and TNCG

This section will introduce some results regarding the NCGs and TNCGs of non-directed graphs which were derived by the author based on Gaschnig's work. We have seen in Section 4.1 how a Neighbors Class Graph (NCG) may be simply represented by replacing the two weighted directed arcs between each pair of vertices by a single, labelled non-directed arc. A single integer (e.g. "[6]") labelled each NCG vertex and was the cardinality of the class it represented. Likewise, a pair of integers (e.g. "2|1") labelled each non-directed arc and stood for the original weightings of the directed arcs it replaced. These are the *connection densities* of the two classes, represented by the vertices, with respect to each other. We shall consider in this section some numerical relationships between these integer labels. The issue of recovering information about the original from its NCG will also be addressed. This is not possible in all cases since the derivation of the NCG is an information-reducing operation. More will be said on this below.

Without loss of generality we shall consider two representative vertices X and Y of an NCG with the single non-directed arc between them and with appropriate positive non-zero integer labels, $m_x, m_y, n_x, n_y > 0$, attached (see Figure 4-8). Note that $m_x = \rho_{xy}$ and $m_y = \rho_{yx}$ are the

Vertex Set of PMS graph in Figure 4-5:

{P1 P2 P3 P4 P5 M1 M2 M3 M4 M5 S1 S2 S3 S4 S5 K1 K2 K3}

Partition by Equal Type equivalence relation:

{P1 P2 P3 P4 P5} {M1 M2 M3 M4 M5} {S1 S2 S3 S4 S5} {K1 K2 K3}

Further partition by Equal Degree equivalence relation
(Subscript "d=n" indicates each vertex in class is of degree n)
$\{P1\ P2\ P3\ P4\ P5\}_{d=1}$ $\{M1\ M2\ M3\ M4\ M5\}_{d=1}$ $\{S1\ S2\ S3\ S4\ S5\}_{d=3}$ $\{K1\ K2\}_{d=4}$ $\{K3\}_{d=3}$

Iterative Splitting to achieve partition by NCER:
(* ⇒ NC-inconsistent class)

STEP 1

{P1 P2 P3 P4 P5}	{M1 M2 M3 M4 M5}	{S1 S2 S3 S4 S5}*	{K1 K2}
			{K3}

STEP 2

{P1 P2 P3 P4 P5}*	{M1 M2 M3 M4 M5}*	{S1 S2 S3 S4}	{K1 K2}
		{S5}	{K3}

STEP 3

{P1 P2 P3 P4}	{M1 M2 M3 M4}	{S1 S2 S3 S4}	{K1 K2}
{P5}	{M5}	{S5}	{K3}

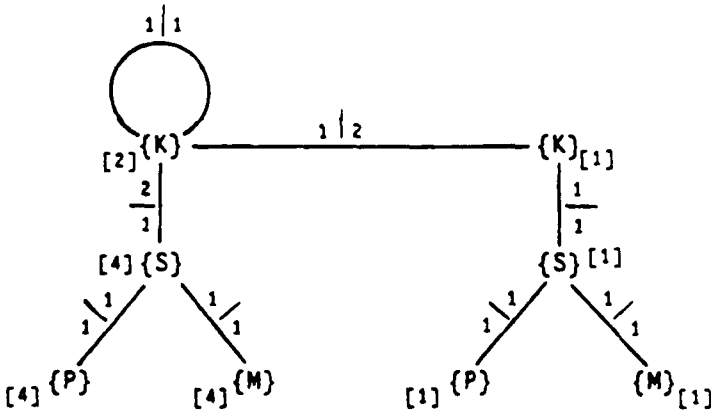
NCER Partition:

Class No.	Class Type	Class
1	P	{P1 P2 P3 P4}
2	P	{P5}
3	M	{M1 M2 M3 M4}
4	M	{M5}
5	S	{S1 S2 S3 S4}
6	S	{S5}
7	K	{K1 K2}
8	K	{K3}

Figure 4-6: Steps of the ETEDS algorithm applied to Figure 4-5

		1 P	2 P	3 M	4 M	5 S	6 S	7 K	8 K
1	P	0	0	0	0	1	0	0	0
2	P	0	0	0	0	0	1	0	0
3	M	0	0	0	0	1	0	0	0
4	M	0	0	0	0	0	1	0	0
5	S	1	0	1	0	0	0	1	0
6	S	0	1	0	1	0	0	0	1
7	K	0	0	0	0	2	0	1	1
8	K	0	0	0	0	0	1	2	0

(a)



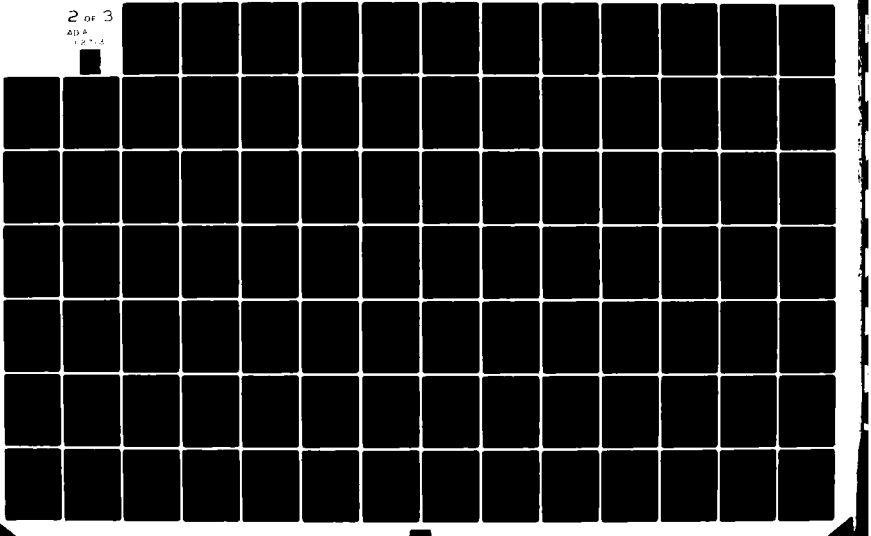
Note: Integers in brackets are the class cardinalities.

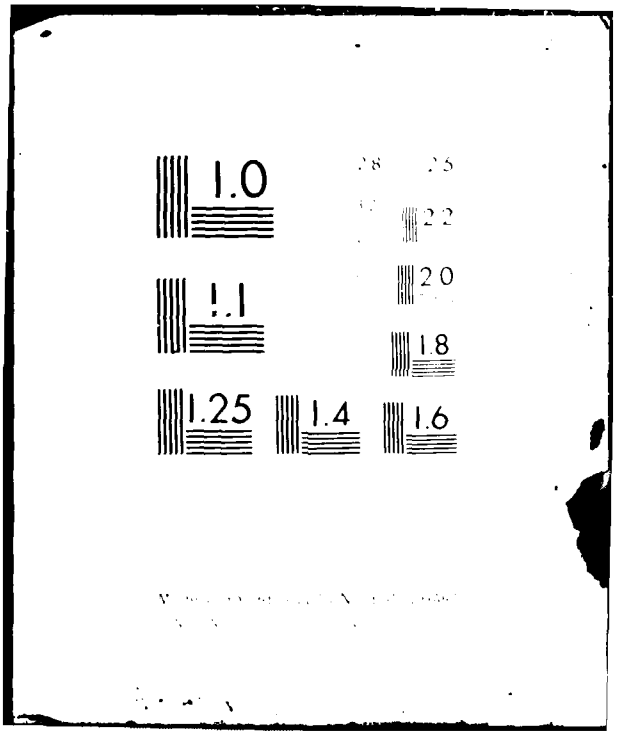
(b)

Figure 4-7: (a) The TNCAM for Figures 4-5 and 4-6
(b) The TNCG defined by the TNCAM above.

AD-A112 713 CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER --ETC F/G 9/2
AUTOMATIC GENERATION OF RELIABILITY FUNCTIONS FOR PROCESSOR-MEM--ETC(U)
FEB 81 V KINI N00014-77-C-0103
UNCLASSIFIED CMU-CS-81-121 NL

2 OF 3
ADA




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2

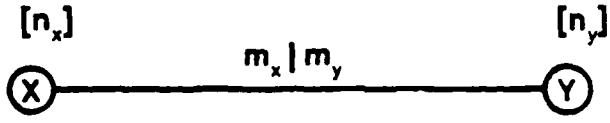


Figure 4-8: A pair of vertices in an NCG.

connection densities. We shall consider this single pair of representative vertices in isolation. Of course, both X and Y may possibly be connected to other neighbor vertices in the NCG but these connections will be symmetric, and results derived for a single pair of vertices X and Y may be applied to all other pairs which contain either of X or Y, and thence to the rest of the graph. The case that X and Y are the same vertex (class) is also considered in Section 4.4.2. In what follows, the symbols X, Y, m_x, m_y, n_x, n_y refer the reader to Figure 4-8. *The results of this section apply in general only to NCGs of non-directed graphs and no attempt has been made to extend them to directed graphs. We attempt to deduce in this section what structure or set of symmetric structures in the original graph, G, caused the appearance of the single arc in its NCG, G', as shown in Figure 4-8. For our purposes,* the NCG and the TNCG are identical except insofar as the latter has type labels for its vertices unlike the former. Therefore, the following results apply to both.

Theorem 4.6: $n_x m_x = n_y m_y$

Proof: The proof is by conservation of arcs. Since each vertex of class X is connected to m_x vertices of class Y, there are a total of $n_x m_x$ arcs incident on vertices of class X from vertices in class Y. Likewise, for class Y there are $n_y m_y$ such arcs incident of vertices of Y from vertices in X. These two must be identical. ∎

4.4.1 Unequal class cardinalities

We shall assume first that $n_x \neq n_y$. This implies from Theorem 4.6 that $m_x \neq m_y$.

Theorem 4.7: If $n_x \neq n_y$ and n_x and n_y are relatively prime then the only possible interconnection graph between the vertices of class X and class Y is a complete bipartite graph.

Proof: From Theorem 4.6 we know that

$$n_x/m_y = n_y/m_x$$

Now n_x, n_y, m_x and m_y are positive integers, therefore this equality implies that n_x and n_y have a common factor. By our assumption, however, they are relatively prime and so this factor can only be unity. Hence, $m_x = n_y$ and $m_y = n_x$ as a result. This result implies that each $x \in X$ is connected to n_y (i.e. all) $y \in Y$. Likewise, as a result, each $y \in Y$ is connected to n_x $x \in X$. Hence, under the conditions of the theorem a complete bipartite graph is the only possible graph joining the vertices of X and Y. ∎

Note that under the conditions of Theorem 4.7 it is possible with the information from Theorem 4.6 to compute any one of the four quantities (two m's and two n's) if all of the other three are known. Likewise, if the ratio of the m's is known, one n calculated from the other and *vice versa*. We shall see in the following section that if the condition $n_x \neq n_y$ is not satisfied then the former calculation will not be possible due to a many to one mapping.

Theorem 4.8: Assume the following notation

Γ_{xy} = GCD(n_x, n_y) (GCD ⇒ greatest common divisor)

φ_{xy} = any factor of Γ_{xy} (including 1 and Γ_{xy})

$n_{px} = n_x / \Gamma_{xy}$, and

$n_{py} = n_y / \Gamma_{xy}$

Then, if $n_x \neq n_y$ and n_x and n_y have a common factor greater than one, then the appearance of a single edge of the form in Figure 4-8 may originate from one of the following kinds of subgraphs in the original graph G(V,E):

1. Complete bipartite graph of n_x and n_y vertices. ($\varphi_{xy} = 1$)
2. Γ_{xy} occurrences of symmetric complete bipartite graphs of n_{px} and n_{py} vertices. ($\varphi_{xy} = \Gamma_{xy}$)
3. or in general, Γ_{xy}/φ_{xy} occurrences of complete bipartite graphs of ($n_{px}\varphi_{xy}$) and ($n_{py}\varphi_{xy}$) vertices. ($1 \leq \varphi_{xy} \leq \Gamma_{xy}$)

Proof: The NCMs of all $x \in X$ (or all $y \in Y$) are identical due to the NCER. Hence, if the single NCG edge in Figure 4-8 was the result of the "collapsing" together of several subunits which were bipartite graphs, all those subunits must have been symmetric. Since n_x and n_y have a common factor the smallest such subunit will have been a bipartite graph of n_{px} and n_{py} vertices. Also, there will be Γ_{xy} such smallest subunits. From Theorem 4.7 this smallest subunit must be a complete bipartite graph since n_{px} and n_{py} are relatively prime. This gives rise to case 2 above. Case 1 will trivially generate a single NCG edge.

Now consider case 3. Symmetry conditions dictate that the number of subunits must be Γ_{xy}/φ_{xy} since the vertices in class X (or class Y) must be evenly divided among the symmetric subunits. In other words, for each symmetric subunit, the number of vertices in their X and Y classes will have to be the same integral multiple (φ_{xy}) of n_{px} and n_{py} respectively. Now from Theorem 4.6 we have the constraint

$$m_x/m_y = n_y/n_x = n_{py}/n_{px}$$

Hence for a subunit which has $n_{px}\varphi_{xy}$ vertices in class X and $n_{py}\varphi_{xy}$ vertices in class Y, m_x and m_y can only take the values $n_{py}\varphi_{xy}$ and $n_{px}\varphi_{xy}$ respectively. This leads to a complete bigraph of $n_{px}\varphi_{xy}$ and $n_{py}\varphi_{xy}$ vertices. ∎

Note that under the conditions of Theorem 4.8 the number of symmetric complete bipartite graphs which produced the single NCG edge will be completely determined by the values of m_x and m_y and, in fact will be GCD(m_x,m_y).

The results of Theorem 4.8 may be used while doing an algorithmic "walk" of an NCG to discover for each edge in the NCG the local structure of the original graph which was reduced by the symmetry detection to that single NCG edge. A special version of these results is embodied in Theorem 4.9 which is used to discover which parts of the NCG correspond to symmetric collections Pendant Tree Subgraphs in the original PMS interconnection graph (see Chapter 5).

4.4.2 Equal class cardinalities

We now consider the case wherein $n_x = n_y$. From Theorem 4.6 we see that $n_x = n_y \Rightarrow m_x = m_y$. It is not possible, however, to solve for the values of the m's since there is no unique solution to the m's in the equation $m_x n_x = m_y n_y$ when $n_x = n_y$. The only observation which can be made is that the connection pattern between the vertices in class X and vertices in class Y will be regular in some sense and constrained by the fact that the degrees of all vertices in both classes are equal (i.e. $m_x = m_y = m$). For instance Figure 4-9 shows cases which "collapse" to an identical edge in the NCG.

An interesting special case is one in which the class X is identically class Y, i.e. there is a self loop on a vertex in the NCG. In this case the connections are between vertices in the same class and symmetry causes them to be cyclic. The connection density in this case specifies the length of the cycle. If the connection density is equal to the cardinality of the class with the self-loop ($m_x = m_y = n_x = n_y = k$) then the self-loop indicates the existence of a k-

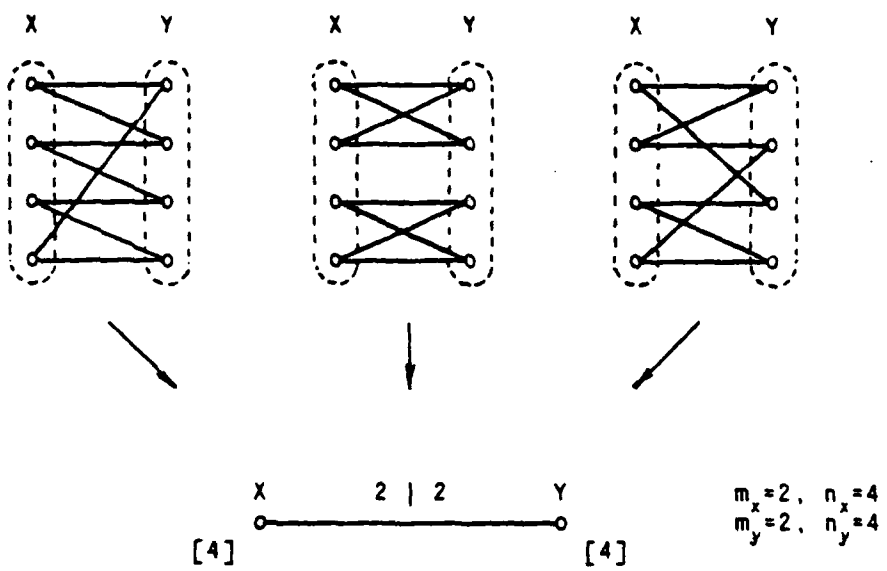


Figure 4-9: Ambiguous origin of single NCG edge when $n_x = n_y$.

clique[13] subgraph in G. If m<n then we have a star polygon with a period of m in G.[14]

4.5 Symmetric trees

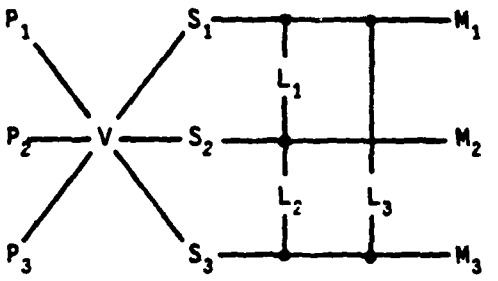
We now consider a special case wherein at least one of the connection densities in Figure 4-8 is unity. Without loss of generality let us assume that $m_x = 1$. Then Theorem 4.6 informs us that the number of vertices in class Y is exactly the number of vertices in class X divided by m_y. Under this condition it will be noted that the bipartite graphs must now be two level trees. In our case, with $m_x = 1$ the roots of these trees are the vertices in class Y and the leaves are the vertices in class X. Also, each vertex (root) in Y has exactly m_y successors or sons in class X. Therefore, the number of trees which "collapsed" to provide the single edge in the NCG is equal to n_y (i.e. the cardinality of the vertex class which holds the root vertices). Hence we have

[13] A *k-clique* is a complete graph on k nodes.

[14] A star polygon is a regular graph which is cyclically connected. It can be completely characterized by an indexed expression which is a function of the degree of each vertex in the graph and the period of the cyclic interconnection (See [Boesch 72]).

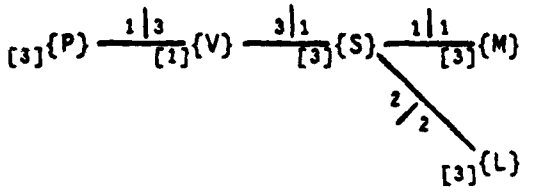
Theorem 4.9: The only condition under which the single NCG edge in Figure 4-8 represents the collapsing of tree subgraphs of G is when at least one of m_x or m_y are identically unity.

The proof of this is obvious. A fact that follows from this is that if $m_x = m_y = 1$ then the only origin in G of the single NCG edge is the collapsing of single edges of G which are $n_x = n_y = n$ in number. Note also that it is possible to have leaves or pendant vertices in G′ which do not correspond to leaves in G. This may be seen by considering the case of Figure 4-10. Theorem 4.9 is made use of in Chapter 5 for discovering specific tree subgraphs of the PMS interconnection graph G. These are then used as a basis for partitioning G in a divide-and-conquer approach.



G(V,E)

SYMMETRY DETECTION



TNCG = G′(V′,E′)

Note: $L_1 \in \{L\}$; {L} is a leaf of G′; L_1 is not a leaf of G.

Figure 4-10: A case where a leaf of G′ is not a leaf of G.

4.6 Conclusion

This section introduced an algorithm to discover symmetries in an unlabelled graph. This algorithm was based on the Neighbors Class Equivalence Relation and gave rise to an auxiliary graph called the Neighbors Class graph. Slight modifications to the algorithm allowed the detection of symmetries in labelled graphs. The identification was drawn between a PMS interconnection graph and a labelled graph and the algorithms were shown to be useful in discovering physical symmetries in a PMS structure. Finally some properties of the NCG were discussed which allowed the deduction from the NCG of the nature of the symmetric subparts of the PMS graph.

In later chapters only symmetric subtrees of a specific kind in the PMS interconnection graph G are used to reduce the amount of computation in the reliability calculation. This is entirely due to the fact that special techniques were developed only for tree structures. However, the information gleaned from the NCG regarding the nature of the symmetries in G may allow the reducing of computation in the case of symmetric instances of other kinds of subgraphs if special techniques for them are developed in the future, or their appearance in PMS structures is sufficiently frequent to warrant special consideration.

Chapter 5
Tree Interconnection Structures

This thesis is an attempt to study the feasibility of generating system reliability functions directly from the actual interconnection topology of a PMS system. Interconnection graphs of tree form were a natural starting point for such an investigation. The most important motive for studying tree interconnection graphs is that PMS structures usually contain input-output subsystems which are connected as trees. For instance, Figure 5-1(a) illustrates the case of a disk storage subsystem, and Figure 5-1(b) shows a terminal controller with its network of terminals. The roots of the two tree-interconnected subgraphs are the $K_{multiplexor.channel}$ and $K_{terminal}$ respectively.

Another obvious, although secondary reason to begin by examining trees is that there exists a large base of efficient algorithms, using trees as data structures, which have been explored and described in the extant literature. During the course of research on this dissertation, however, it was found that most of these algorithms were inapplicable except to do minor subtasks in the reliability calculation process which was contemplated. This situation is largely due to the fact that the reliability calculation problem is combinatorial whereas extant algorithms address much simpler and more basic problems in manipulating tree data structures. Algorithms derived for such classes of problems as the generation of spanning trees of graphs, shortest path problems, tree searching etc. appeared irrelevant to the particular task at hand.

> ***Definition 5.1:*** A Pendant Tree Subgraph (PTS), T, of a PMS structure interconnection graph, G, is a maximal rooted tree subgraph such that the root vertex of T is an articulation vertex of G and the simple path, p_{xy}, between any pair of vertices v_x and v_y in T, is unique in G. The root vertex of a PTS shall be termed an interface vertex.

It may be seen that the two trees in Figure 5-1 are indeed PTSs and that their respective root vertices are "interface" vertices to the rest of the interconnection graph. In order for any component in a PTS to be useful to other subsystems not in that PTS, the component

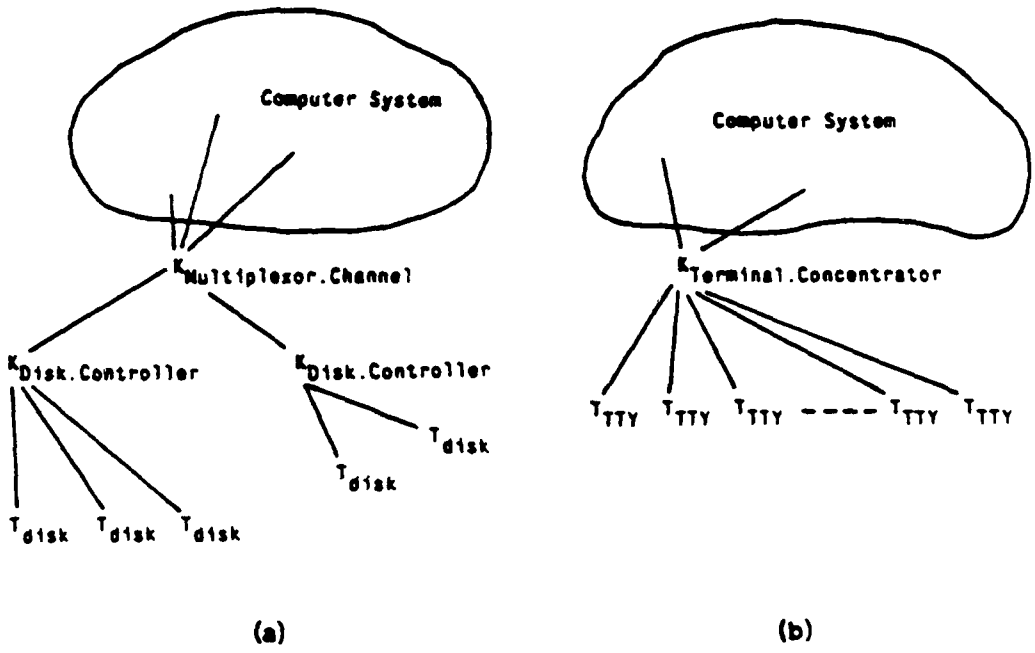


Figure 5-1: Examples of Pendant Tree Subgraphs

represented by the root vertex of that PTS must be functional. This is so that a route may exist for information to flow between the PTS subsystem and other parts of the PMS structure. Also since the interface vertex is an articulation vertex of the PMS graph the PTS reliability may be considered separately from the main graph. This is due to the sets of components in the PTS being disjoint from the rest of the graph and the assumption of independence of failure behavior. The functionality of the PTS when viewed from the rest of the system is dependent on whether the interface vertex is functioning or not. The algorithm developed below for calculating the reliability of tree structures depends on this fact.

Figure 5-2 shows the portion of ADVISER which is discussed in this chapter. In Section 5.1 we shall introduce and discuss an algorithm used for detecting Pendant Tree Subgraphs in the graphs of PMS structures. The algorithm will employ symmetry information gained by the use of the symmetry detection algorithms of Chapter 4. The following section, Section 5.2, will

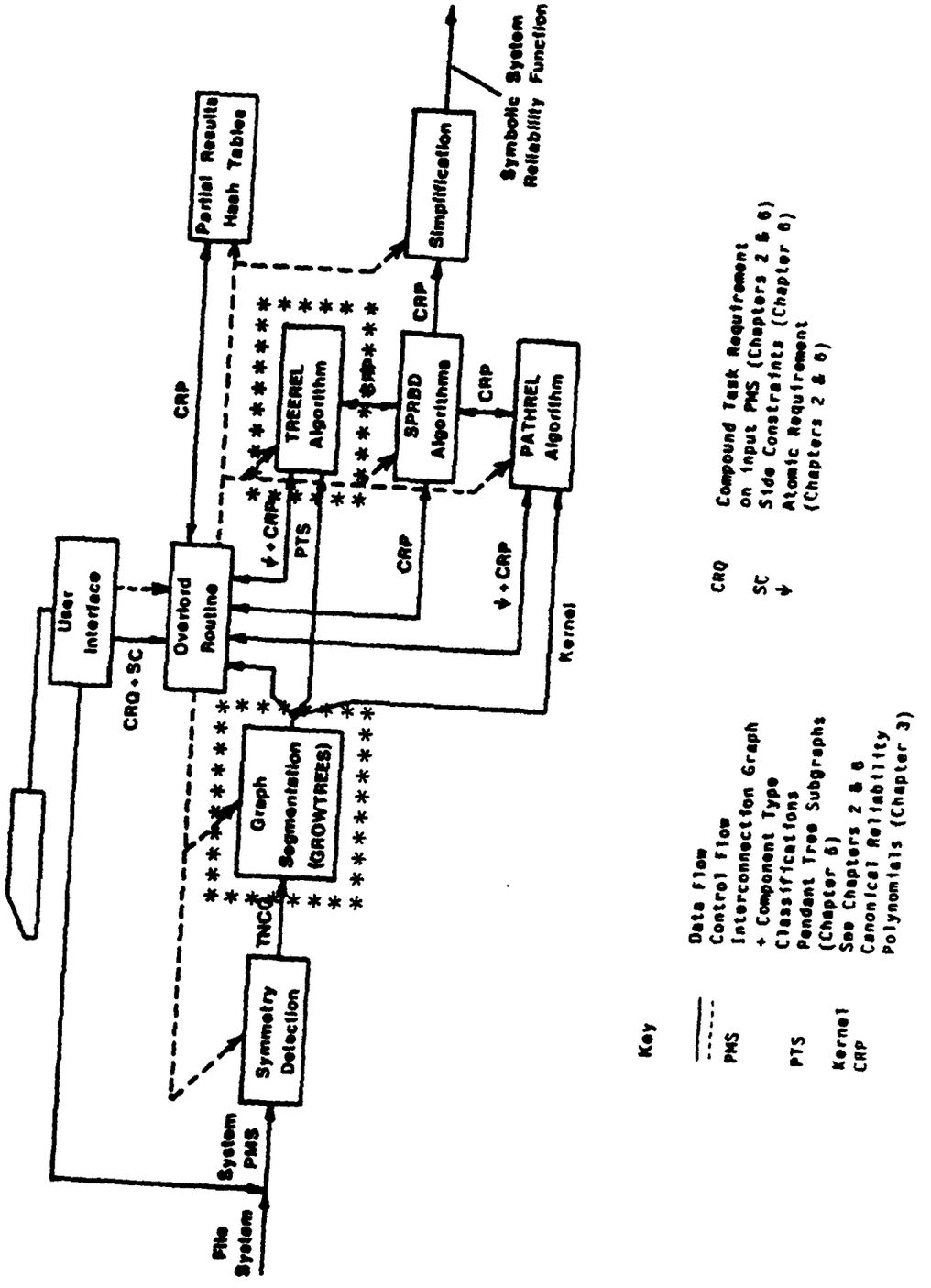


Figure 5-2: The portion of the ADVISER structure discussed in Chapter 5.
Also see Page 18.

describe TREEREL algorithm for generating the symbolic reliability function for a Pendant Tree Subgraph given a boolean requirements expression. The chapter will conclude with a note on current known deficiencies of the TREEREL algorithm.

5.1 Generation of Pendant Tree Subgraphs (PTS)

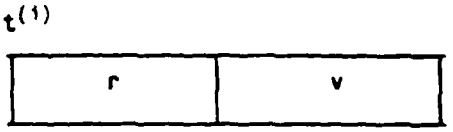
The reader will recall that the result of the symmetry detection process described in Chapter 4 is a Typed Neighbors Class Graph (TNCG) which has as its vertices the equivalence classes resulting from the ETEDS algorithm. These vertices are labelled with the type and cardinality of the respective classes. The edge between any two vertices is labelled with the pair of connection densities of the two classes with respect to each other. Theorem 4.9 in Chapter 4 showed that in order for a subgraph of the NCG, G′, to have originated from a tree in G(V,E), the connection densities in the direction of the root should always be unity. Capitalizing on this result and the additional obvious result that a pendant or leaf vertex in the original graph G(V,E) must imply a leaf vertex in the NCG,[15] an algorithm may be generated to discover the pendant tree subgraphs of G. We shall assume that the PMS interconnection graph G is not a tree graph itself although it may have PTSs. Appendix A examines the consequence of removing this restriction.

The algorithm proceeds by first collecting the set of pendant vertices in the NCG, G′, of G. These leaves of G′ represent those vertex classes which have only one neighbor. From this set of pendant vertices, those vertices are deleted whose connection density to their single neighbor class is greater than one since they obviously cannot represent leaves of a PTS in G. The set of remaining pendant vertices of G′, which we shall denote V'_p, does represent the set of leaves of the PTSs in G. We shall metaphorically term the members of V'_p *germinal trees*, and the process of constructing the PTSs from them as *growing trees*. The reason for this metaphor will become apparent.

Each member of V'_p is mapped into a data structure t as shown in Figure 5-3. The field t_r will hold a single vertex of NCG G′. The field t_v will hold a *set* of vertices taken from G′. Let T'_{pt} be the set $\{t^{(i)}\}$, $|\{T'_{pt}\}| = |\{V'_p\}|$, where each $t^{(i)} \in T'_{pt}$ corresponds uniquely to a $v'_i \in V'_p$. The algorithm starts by assigning $t_r^{(i)} \leftarrow v'_i$ and $t_v^{(i)} \leftarrow \{v'_i\}$. This initializes the root $t_r^{(i)}$ and the vertex set $t_v^{(i)}$ of the i^{th} germinal tree, to the NCG leaf vertex v'_i.

---

[15] although not *vice versa* in general since the connection density of that pendant NCG vertex to its neighbors could be greater than unity



Key

r -- root vertex, $t_r^{(1)}$

v -- vertex set, $t_v^{(1)}$

Figure 5-3: Data structure for germinal trees

The algorithm then iterates in two passes. It first examines each $t^{(i)}$ in turn. If the root vertex $t_r^{(i)}$ of the i^{th} germinal tree has *exactly one* neighbor v'' in G' *which is not already contained in the vertex set* $t_v^{(i)}$ *of any of the germinal trees* and, in addition, the connection density of $t_r^{(i)}$ with respect to v'' is *exactly* one then $t_r^{(i)} \leftarrow v''$ and $t_v^{(i)} \leftarrow t_v^{(i)} \cup v''$. These conditions have to be imposed since G is a non-directed graph. If at any stage, $t_r^{(i)}$ has several neighbors, its connection density to all of whom is unity, then there is no way to know which of them is at a higher level in the PTS. Some of them may be "brother" vertices at the same level of the PTS as $t_r^{(i)}$. Thus, any one of them could, from the point of view of $t_r^{(i)}$, be the next higher root to be added. Of course, those neighbors of $t_r^{(i)}$ in whose direction the connection density is greater than one can never be added to $t^{(i)}$ and will most likely be part of another germinal tree. Hence the growing process of the germinal tree $t^{(i)}$ may have to wait through several iterations of the algorithm until all but one of those neighbors with unit connection density has been consumed by other germinal trees growing upward from their leaves. Or else, if the algorithm terminates before this happens, then $t_r^{(i)}$ is the *de facto* root, or interface vertex, of a PTS.

At the end of this first pass, each germinal tree may have "grown" by one more vertex toward the root of the PTS of which it is a subtree. The next pass over the set of $t^{(i)}$ checks to see if there are any two germinal trees that have the same root vertex. If so, these are merged or coalesced into one germinal tree i.e. $\exists i,j$ such that $t_r^{(i)} = t_r^{(j)}$ then $t_v^{(i)} \leftarrow t_v^{(i)} \cup t_v^{(j)}$ and $t_v^{(j)}$ is deleted.

Note that if, at any stage of the iteration, the root $r_x = t_r^{(i)}$ of any germinal tree i at that stage has itself for a neighbor vertex, then no more vertices can be added on to that germinal tree at any further stage (This is also in accord with Theorem 4.9 in Chapter 4). Such a tree is then said to have stopped growing and is no longer considered in further passes except for merging into other germinal trees that have grown toward the same root vertex.

When there is no further change during this iterative process i.e. when no more neighbor vertices can be added and no more germinal trees can be coalesced, the process terminates. The resulting trees are the PTSs of the NCG G′. It will be noted that due to the symmetries detected during the construction of G′, the cardinality of the class represented by the root vertex of any PTS of G′ gives the number of physically symmetric PTSs of G which are represented by that PTS of G′. This fact has the obvious consequence that since symmetric PTSs of G have been recognized and localized, the reliability function of one of a set of symmetric trees will have the same form as all the others in the set, thus effecting some computational savings. This issue is further discussed in Chapter 6. Algorithm GROW is described below

Algorithm GROW

Terminology:

- The symbol ρ'_1 will be used whenever there is exactly *one* neighbor, of a given vertex v_1, under consideration and will stand for the connection density of v_i with respect to that one neighbor.

- The symbol v_{un} will be used whenever there is exactly one neighbor of a given vertex under consideration and will represent that *single neighbor* (un⇒unique neighbor).

- The set V_p will be assumed to be initialized as follows:

$$V_p \leftarrow \{v'_i | v'_i \in V', d(v'_i) = 1, \rho'_i = 1\}$$

- The function "MarkComplete" causes a germinal tree to be labelled as not capable of further growth thus removing it from consideration during the *further* iterations of the algorithm. The function "MarkDead" removes a germinal tree from further consideration once it has been coalesced with some other germinal tree. The functions "MarkedAsComplete" and "MarkedAsDead" check to see whether the germinal tree given as their parameter has been respectively marked as complete or dead.

```
Procedure GROW
begin

for i from 1 to |V_p|
                          Comment |V_p|⇒cardinality of set V_p;
do   t_r^(i) ← v'_i;   t_v^(i) ← {v'_i}  od;

               Comment v'_i∈V_p, t^(i)∈T_pt;

changes←true;

while changes
do
BEGIN

changes←false;
for i from 1 to |T_pt|
do
   if not MarkedAsDead(t^(i) )
   then
     if not MarkedAsComplete(t^(i) )
     then
        neighbors ← GetNCM(t_r^(i));

        if t_r^(i) ∈ neighbors
        then MarkComplete(t^(i) )
        else
             neighbors ← neighbors - neighbors ∩ t_v^(i) ;
             if |neighbors|=1
             then
                  if ρ'_i =1
                  then
                       t_r^(i) ← v_un ∈ neighbors;
                     Comment single neighbor remaining in "neighbors"

                       t_r^(j) ← t_r^(j) ∪  v_un
                  else
                       MarkComplete(t^(i) )
                  fi
             fi
        fi
     fi
   fi
od;
```

```
Comment now merge germinal trees that have overlapped at the root;

for i from 1 to |T_pt|
do
if not MarkedAsDead(t^(i) )
then
     for j from i+1 to |T_pt|
     do
     if not MarkedAsDead(t^(j) )
     then
          if t_r^(i) = t_r^(j)
          then
               t_v^(i) ← t_v^(i) ∪ t_v^(j) ;
               if MarkedAsComplete(t^(j) )
               then MarkComplete(t^(i) )
               fi;
               MarkDead(t^(j) );
               changes←true
          fi
     fi
     od
fi
od

END

end;     Comment end of Procedure GROW;
```

5.2 Generation of Reliability Functions for PTSs

Previous sections in this chapter discussed the process of recognition of Pendant Tree Subgraphs (PTSs) of a PMS interconnection graph. We now approach the question of generating reliability functions for such tree structures. The methods developed in this chapter for this task are used to generate partial results regarding such PTSs in the overall interconnection structure. Such partial results along with others are operated upon to produce the final result which is the reliability function of the entire PMS structure. The reader will recall from Section 2.2.1.3 that an atomic requirement on a PMS structure is a clause of the form "at least N of X", represented symbolically by $\psi(N,X)$, where N is an integer and X is a distinct type of component in the structure. In the following we shall initially indicate how reliability functions for PTSs may be derived for such atomic requirements and then generalize the result to a Boolean function on atomic requirements, i.e. compound requirements.

The algorithm starts on the distinguished, or root, vertex of a PTS of G, the PMS interconnection graph.[16] This is an articulation vertex of G(V,E) and is termed an interface vertex. The component represented by it must be functional in order for other functioning vertices in the tree to be able to satisfy the Communication Axiom (see Sections 2.1 and 6.6.1) for system reliability. To introduce the algorithm we shall consider a complete m-ary tree of infinite extent and composed of homogeneous vertices. In other words all the components represented by the vertices of the tree are of exactly the same component type, say X. We shall, in addition, assume an atomic requirement of $\psi(N,X)$. Starting from the root vertex of the PTS the algorithm recursively descends into the tree keeping count of how many vertices of type X have been encountered thus far. Since at least N components of type X are required to be functional, a functional state of the tree is found as soon as N such vertices have been encountered. There is then no need to descend farther into the tree since the requirement has been met and the states (working or failed) of components lower in the tree are not of consequence. The algorithm then accounts for this functional state in the partial result thus far accumulated (we shall presently describe what is meant by accumulation of results) and backs up to try the next possibility. In this sense the procedure is exhaustive but only functional states of the PTS are examined. The algorithm will be described in detail below.

It is clear that in our example of a homogeneous tree the descent will encompass no more than N levels of the tree including and starting from the root vertex. The following question then comes to mind; If the homogeneous tree is N + 1 levels deep, say, then what of the vertices that are the leaves of the tree at level N + 1? Since they are all also of the required type X, may they not also contribute to some functional state? On a little reflection it is apparent that the constraint that excludes such possibilities is that all communication must flow through the root vertex r into the rest of G(V,E). Thus, for any functioning vertex v at level l to be part of a functional state, all vertices along the path of l-1 edges from v to r must be functional. In our instance, all of these vertices are of the required type. Hence if $l \leq N$ then each vertex of the tree will appear in one or more functional states. However, if $l > N$ then there will be at least one vertex whose functioning or non-functioning is irrelevant to the functioning of the tree structure.[17]

---

[16] This is in contrast to the Algorithm GROW which discovers PTSs of the NCG, G', of G thereby indicating symmetric PTSs of G.

[17] The concept of relevancy here is used in the sense of Barlow and Proschan [Barlow 75a]

5.2.1 The TREEREL Algorithm

The TREEREL algorithm uses the notion of the compositions of an integer into some number of parts (for example see [Nijenhuis 78]).

Definition 5.2: A k-composition of a positive integer n is an ordered tuple of k integer parts $p_i \geq 0$, i = 1,...,k, such that $\Sigma_i p_i = n$.

This is to be contrasted with the following definition:

Definition 5.3: A k-partition of a positive integer n is an unordered tuple of k integers parts $p_i > 0$, i = 1,...,k, such that $\Sigma_i p_i = n$.

---

p_1	p_2	p_3	p_4	
3	0	0	0	
2	1	0	0	
2	0	1	0	
2	0	0	1	
1	2	0	0	
1	1	1	0	
1	1	0	1	
1	0	2	0	
1	0	1	1	
1	0	0	2	
0	3	0	0	
0	2	1	0	
0	2	0	1	
0	1	2	0	
0	1	1	1	
0	1	0	2	
0	0	3	0	
0	0	2	1	6=4+1+1
0	0	1	2	6=3+2+1
0	0	0	3	6=2+2+2
	(a)			(b)

Figure 5-4: (a) All the 4-compositions of the integer 3.
(b) All 3-partitions of the integer 6.

At any depth, within the tree operated upon by a recursive incarnation of the PTREE procedure of algorithm TREEREL, let n be the number of required-type components remaining to be found to satisfy the atomic requirement on the tree. Let the root vertex of the subtree currently being studied be r'_0. Furthermore, let r'_0 have m sons $r'_1,...,r'_m$. The algorithm first examines the root vertex r'_0 for its component type. If it is the required type then the

number of components of the required type remaining to be found is decremented by one. The algorithm then proceeds to sequence through all the m-compositions of the integer n (or n-1 if the vertex r_0' was of the required type). For each such m-composition the algorithm is called recursively on each son of r_0', r_i', $i = 1,...,m$, with the parameter p_i as the number of required-type components to be found in the subtree whose root vertex is r_i'. Here p_i is the i^{th} integer part of the m-composition.

For each m-composition of n the values (canonical reliability polynomials) returned by the recursive calls of the PTREE procedure on each of the subtrees r_i' are SMERGEd. The SMERGE procedure (Chapter 3) effects a conjunction of the probabilities represented by the reliability functions returned by the various recursive calls. After all the m-compositions of n have been examined, their individual SMERGE results are then PMERGEd together. The PMERGE operation (Chapter 3) produces a reliability function which represents the disjunction of the probabilities represented by the results of the SMERGE operations. Finally, if the results of this PMERGE operation were non-null, i.e. the subtree rooted on r_0' was able to meet the requirement of n, then the reliability of r_0' is SMERGEd into the results of the PMERGE operation[18]. The results of this final SMERGE operation, if it is invoked, are returned as the value of the current incarnation of procedure TREEREL operating on r_0'. Otherwise a null result is returned indicating that no functional states could be found.

It is evident that for some subtrees in a PTS which is not homogeneous (i.e. components within it are of different types) not all the compositions of the integer requirement over the number of sons of the root of the subtree, will produce fruitful results. In other words, some requirements on a particular component type may be greater than the number of components of that type available in a given subtree. This implies that that subtree can never be functional under that particular requirement. In order to decide whether or not a given requirement can be met by a subtree it is necessary to know beforehand how many components of the required type are available within it.

> ***Definition 5.4:*** Let r be the root of some subtree t(V,E) within a PTS and let x be the required component-type. Then the required resource supply (RRS) σ_T, of the tree t with respect to x is defined as
>
> $$\sigma_T(r,x) = |V'| \text{ such that } V' \subseteq V,\ \forall v \in V' \text{ type}(v) = x$$

---

[18] This is equivalent to stating that regardless of what combinations of components in the subtrees $r_1',...,r_m'$ were chosen to satisfy the requirements, the root vertex r_0' will always have to be functional for those combinations to be useful

The subscript T on the σ indicates the applicability of the definition to rooted tree subgraphs of the PTS. Such a tree t is represented by its root r and thus r is a parameter of σ_T.

The quantity $\sigma_T(r,x)$ of each vertex r in the PTS is gathered in an O(N) post-order traversal of the PTS prior to running the PTREE algorithm.

Definition 5.5: Let r_0' be the root of some subtree t of the PTS which is being operated upon by a recursive incarnation of Procedure PTREE. Let r_i', $i = 1,...,m$, be the m immediate successors vertices of r_0'. Let x be the required component type and n be the remnant of the integer requirement to be applied toward the subtrees of r_0' after subtracting one in case type(r_0') = x. Then an m-composition $c \equiv \{p_i\}$ of n over the r_i', $i = 1,...,m$, $\Sigma_i p_i = n$, is said to be a feasible composition (or c is feasible) iff

$$\sigma_T(r_i',x) \geq p_i,\ i = 1,...,m$$

A composition c is said to be infeasible iff it is not feasible.

During each recursive call to the PTREE algorithm on the root vertex r_0' of some subtree of the PTS, the m-compositions of n are generated as described above. A composition is considered, and recursive calls to PTREE on r_i' are initiated, only if the composition is feasible. The procedure NEXTFCOM, described below, is used to generate the next feasible composition at each step.

A further refinement of the algorithm is possible and was made in the following way. For a given subtree rooted on some r_0', and for a given required component type x, in general,

$$\sigma_T(r_i',x) \geq 0,\ i = 1,...,m$$

Now if $\exists\ j \in i = 1,...,m$, such that $\sigma_T(r_j',x) = 0$ then any composition which has $p_j > 0$ will be infeasible. Thus, when generating compositions we need only consider those r_i' such that $\sigma_T(r_i',x) > 0$. In such a case the number of parts in the composition will be equal to the number of r_i' whose $\sigma_T(r_i',x) > 0$. In view of this, an additional O(N) post-order traversal is made over the PTS prior to the initiation of the PTREE algorithm to prune subtrees whose σ_T is equal to zero. Furthermore, the remaining subtrees of each vertex of the PTS are ordered in ascending order of their σ_T. This allows the generation of the compositions to be started directly at the first feasible composition and is reflected in the procedures NEXTFCOM and PTREE below.

There are, thus, three stages in the TREEREL algorithm, namely

- Compute σ_T for all vertices in the PTS for the required component type x.

- Prune the subtrees with $\sigma_T = 0$ (in other words we effectively reduce m) and reorder the remaining subtrees in ascending order of their σ_T.

- Call the PTREE procedure on the root vertex of the PTS with the integer requirement n.

The value returned by the PTREE algorithm is the canonical reliability polynomial (CRP) of the PTS under the atomic requirement $\psi(n,x)$. Shown below are the four procedures that comprise Algorithm TREEREL.

Algorithm TREEREL

Terminology:

- succ(r,i) is a function which returns the i^{th} immediate successor vertex of r.

- nsucc(r) is a function which returns the number of immediate successors of vertex r.

- SMERGE and PMERGE are algorithms described in Chapter 3. They respectively return the symbolic reliability function of the event which is the conjunction or disjunction of the two events represented by their parameters.

- Procedure NEXTFCOM is a modified version of Algorithm NEXTCOM in [Nijenhuis 78]. The latter generates all m-compositions of the integer n in the order as shown in Figure 5-4(a). The former only generates those m-compositions which are feasible.

- The names of the parameters to TREEREL are reasonably self-explanatory: "ptsroot" is the root vertex of the PTS for which the reliability function is to be generated, "reqtype" is the required resource type of the atomic requirement, and "reqment" is the integer requirement of the atomic requirement.

```
Procedure TREEREL (ptsroot,reqtype,reqment)
begin
```

```
Comment first declare four procedures which are used by TREEREL;

  Procedure FINDSUPPLY (r,x)
                  Comment finds σT of subtrees of the given PTS;
  begin
    integer supply;
  if type(r)=x then supply←supply+1 fi;
  for i from 1 to nsucc(r)
  do
     supply←supply+FINDSUPPLY(succ(r,i),x)
  od;
  σT(r,x)←supply;
  return supply
  end;     Comment end of Procedure FINDSUPPLY;


  Procedure PRUNEANDSORT (r,x)
          Comment prunes those subtrees with σT=0 and rearranges;
  begin
  for i from 1 to nsucc(r)
  do
    if σT(succ(r,i),x)=0
    then (prune subtree rooted on succ(r,i))
    fi
  od;
  (quicksort the remaining successors of r
  into ascending order of σT);
  for j from 1 to nsucc(r)
  do PRUNEANDSORT(succ(r,j),x) od
  end;     Comment end of Procedure PRUNEANDSORT;
```

```
Procedure NEXTFCOM (
                reference integer array com[1:m],
                reference integer array sigmat[1:m],
                integer n,
                integer m
                  )
       Comment generates next feasible m-composition of n given
          σ_T vector in sigmat[1:m]. Previous feasible m-composition
          of n resides in com[1:m];
begin
  integer h, t;
  label newcomposition;
do
  begin
  if com[m]=n then return false fi;
  (h←com[i] where i is the smallest i = 1,...,m such that com[i]>0);
  t←com[h]; com[h]←0;
  com[1]←0;
  com[h+1]←com[h+1]+1
  end
until
  newcomposition:
  begin
  for i from 1 to m
  do
     if sigmat[i] < com[i]
     then leave newcomposition with false
  od;
  leave newcomposition with true
  end
od
return true
end;     Comment end of Procedure NEXTFCOM;


Procedure PTREE (r,n,x)
     Comment recursively computes symbolic reliability function of
             subtree rooted on r with requirement of ψ(n,x);
begin
  if n > σ_T(r,x) then return nil fi;
  if type(r) = x then n←n-1 fi;
  if n = 0
  then
      return R_r   Comment symbolic reliability of
                                  component represented by r;
  else
      begin            Comment MAIN BLOCK
             integer tosatisfy;
             integer array com[1:nsucc(r)],sigmat[1:nsucc(r)];
             reliability function value, tempvalue;
       tosatisfy←n;     Comment need to find n components of type x;
```

```
        Comment initialize for first feasible composition
        for i from 1 to nsucc(r)
        do
           sigmat[i]←σ_T(succ(r,i),x);
           if sigmat[i] < tosatisfy
           then com[i]←sigmat[i];
                tosatisfy←tosatisfy-sigmat[i]
           else com[i]←tosatisfy;
                tosatisfy←0  fi
        od;

        value←nil;

        Comment for each feasible composition returned from NEXTFCOM
              generate partial result for each subtree and accumulate;
        do
          tempvalue←nil;
          for i from 1 to nsucc(r)
          do tempvalue←SMERGE(tempvalue,PTREE(succ(r,i),com[i],x)) od;
          value←PMERGE(tempvalue,value)
        until not NEXTFCOM(com,sigmat,n,nsucc(r))
        od;


    Comment if accumulated partial result is not nil then
       root vertex of current subtree will also be required,
       therefore, SMERGE in its reliability function;
        if value = nil
        then return nil
        else return SMERGE(value,R_r) fi

        end                    Comment end of MAIN BLOCK
    fi

end;   Comment end of procedure PTREE;
Comment Declarations for Procedure TREEREL end here;


Comment The code for the Procedure TREEREL begins here;

if FINDSUPPLY(ptsroot,reqtype) = 0 or reqment = 0
then return nil fi;

PRUNEANDSORT(ptsroot,reqtype);

return PTREE(ptsroot,reqment,reqtype)

end; Comment end of algorithm TREEREL;
```

5.2.2 Analysis of Procedure PTREE

The analysis of the PTREE procedure is intractable for non-homogeneous trees and incomplete[19] m-ary trees. We shall use the infinite homogeneous complete m-ary tree to investigate the nature of the algorithm. This kind of tree turns out to be a worst case example since it offers the maximum number of possible functional states for a given requirement. There is little doubt at the outset, however, that the algorithm is combinatorial in nature. Fortunately, the sizes of trees expected to be dealt with by the ADVISER program is small; in the order of 20 vertices or so at most for extant multiprocessor interconnection structures. Moreover, in practice, the PTSs of a PMS graph are more likely to be non-homogeneous and incomplete m-ary trees. Thus at each level, many of the compositions to be examined by Procedure PTREE will turn out to be infeasible and therefore will not even be returned for consideration by Procedure NEXTFCOM. This will speed up the algorithm on the average. At any rate, concerns which arose early during the course of this work with regard to the complexity of algorithm TREEREL were found in practice to be misplaced. The ADVISER program expends the largest percentage of its computation time in the PMERGE and SMERGE algorithms during the execution of the OVERLORD routine (see Chapters 3, 6 and 7 respectively).

We shall ignore the fact the pruning and sorting of subtrees is done before PTREE is invoked since this will not happen for our worst case example. We approach the analysis of the PTREE procedure by noting some facts in regard to all the possible m-compositions of an integer. Specifically, consider the table of all possible 4-compositions of the integer 3 (i.e. $m = 4$, $n = 3$) in Figure 5-4(a) and focus attention on the first column of figures (which contains the values of the part p_1 of each composition). It will be noted that the number of times the integer $(n-k)$, $k = 0,1,...,n$, $n = 3$, appears in the first of the four columns is equal to the number of all possible $(m-1)$-compositions of the integer k. This number is

$$\binom{(n-k)+m-2}{m-2} \tag{5.1}$$

Also note that this is true of all of the other columns of Figure 5-4. All the other columns are in addition, just permutations of the first column. Using the terminology above, Procedure PTREE is called recursively on vertex r_1' with the integer p_1 as the requirement, unless $p_1 = 0$. Thus, the number of times Procedure PTREE is called on vertex r_1' with an integer requirement of $(n-k)$ is given by expression (5.1) above, except when $k = n$. This is also true of calls of PTREE on all the other successor vertices of r_0', i.e. $r_2', r_3', ..., r_m'$.

[19] Incomplete trees are described in [Knuth 75b], Pg.401.

Let us define $\omega(n)$ to be the work done by the PTREE algorithm in traversing, to the necessary depth, an infinite, homogeneous and complete m-ary tree with an initial integer requirement of n. We shall posit the initial condition to be $\omega(1) = 1$. In other words, one unit of work is done by each recursive call to PTREE. Then $\omega(n)$ also represents the total number of recursive calls to PTREE under these conditions and we shall consider this to be a measure of the time complexity of the algorithm.

In our idealized homogeneous tree, upon entry into a subtree by Procedure PTREE, the requirement n will always be decremented by one since the root of any subtree is always a component of the required type (see the pseudo-code for Procedure PTREE above). Hence, the m-compositions computed are of the integer (n-1). Thus the total work done on the subtree rooted on r_1' is given by the sum:

$$\Sigma_j \, (\#\text{ times integer } (n-1)-j \text{ appears in column 1}) * \omega(n-1-j)$$

except that occurrences of 0 in the column are ignored. This sum then becomes:

$$\Sigma_{j=0}^{n-2} \binom{j+(m-1)-1}{(m-1)-1} \omega(n-1-j)$$

However, there are m-1 more columns which, except for having their elements permuted, are identical to the first column. Furthermore, we must add one to represent the call of the Procedure PTREE on the root of the subtree, r_0' itself. Hence, we may finally write the expression for $\omega(n)$ as

$$\omega(n) = 1 + m \Sigma_{j=0}^{n-2} \binom{j+m-2}{m-2} \omega(n-j-1) \qquad (5.2)$$

Equation (5.2) is an n^{th} order difference equation in n (m is a constant and completely characterizes the infinite, homogeneous, complete m-ary tree of our example). The problem of obtaining a closed form solution for $\omega(n)$ appears intractable. However, the first five terms of the series $\omega(n)$, $n = 1,2,3,...$ are shown in Figure 5-5. It is clear that $\omega(n)$ grows as $O(m^{n-1})$. This is an interesting result insofar as it implies that for a fixed n the reliability calculation procedure TREEREL is roughly polynomial in the number of components in the tree. However, for fixed m, the algorithm is exponential in n, the requirement integer. It would be expected, therefore, that more complex[20] requirements should affect the computation time

[20] The complexity of requirements may be increased by taking one of three actions, namely (i) introduce more terms into the boolean requirement expression, (ii) make each of the atomic requirements require closer to half of the available components of that required type in the structure. Since the number of functional possibilities for each atomic requirement is given by a binomial coefficient, this action increases the number of possibilities to be considered, and (iii) introduce more disjunctions into the boolean expression as opposed to conjunctions. Since the disjunctions are inclusive-ORs the number of cases to be considered can multiply rapidly.

more substantially than more complex interconnection structures. This correlation has indeed been borne out by experience in using ADVISER. The observation seems also to apply roughly to the PMS reliability calculation problem addressed in this thesis as a whole.

$$\omega(1) = 1$$

$$\omega(2) = 1 + m\{\binom{m-2}{m-2}\omega(1)\} = m+1$$

$$\omega(3) = 1 + m\{\ \omega(2) + (m-1)\omega(1)\ \}$$
$$= 2m^2 + 1$$

$$\omega(4) = 1 + m\{\ \omega(3) + (m-1)\omega(2) + m(m-1)/2!\ \omega(1)\}$$
$$= 7/2\ m^3 - 1/2\ m^2 + 1$$

$$\omega(5) = 1 + m\{\ \omega(4) + (m-1)\omega(3) + m(m-1)/2!\ \omega(2) + (m+1)m(m-1)/3!\ \omega(1)\}$$
$$= 37/6\ m^4 - 15/6\ m^3 + 2/6\ m^2 + 1$$

Figure 5-5: First five terms of $\omega(n)$

5.2.3 Extension of TREEREL to compound requirements

Previous sections were concerned with the generation of reliability functions for PTSs on the basis of atomic requirements. In this section we shall extend those techniques to the non-atomic requirements. These are Boolean functions on the atomic requirements, as described earlier in Chapter 2, wherein the individual atoms may have different integer requirements and required types. An example of such a compound requirement is

$$\psi(5,\text{Processor}) \wedge (\psi(3,\text{M.primary}) \vee (\psi(1,\text{M.disk}) \wedge \psi(2,\text{M.primary}))) \qquad (5.3)$$

Such a Boolean function may be represented by its parse tree (see Figure 5-6). The latter, which we shall term the requirements tree, is a binary tree wherein the leaf vertices represent the atomic requirements and the non-terminal vertices represent the Boolean operators AND and OR. The extension capable of handling such a compound requirement is simple. It essentially involves a treewalk of the requirements tree with an invocation of Algorithm TREEREL, on the given PTS, performed with the atomic requirement at each leaf of the requirements tree. The extended version of the TREEREL algorithm is shown below

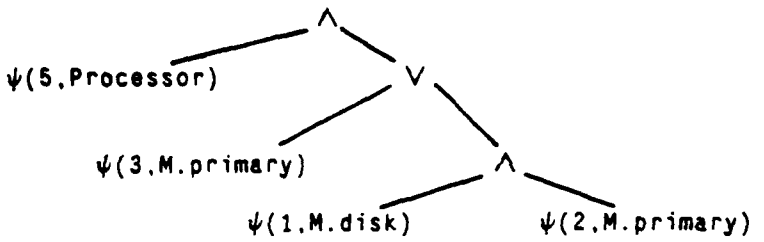


Figure 5-6: Parse tree of requirement expression (5.3).

Algorithm EXTREEREL

Terminology: As in Algorithm TREEREL with the following additions

- "rqtvertex" is a vertex of the requirements tree on which an incarnation of the Procedure EXTREEREL has been invoked. The algorithm begins by call the initial incarnation of EXTREEREL on the root of the requirements tree.

- ISOP is a function which returns TRUE if its parameter is an operator (i.e. a leaf) vertex of the requirements tree.

- GETOP is a function which returns the operator represented by the non-terminal vertex, of the requirements tree, which is its parameter

- GETREQMNT and GETREQTYP are functions which return, respectively, the integer requirement and the required type of the atomic requirement represented by the requirements tree vertex which is passed to them as a parameter.

- LEFTSON and RIGHTSON are functions which return the successor vertices of any vertex in the binary requirements tree.

```
Procedure EXTREEREL (rqtvertex,ptsroot)
begin

     reliability function leftr, rightr;

if
    ISOP(rqtvertex)
then
    leftr ← EXTREEREL( LEFTSON(rqtvertex), ptsroot);
    rightr ← EXTREEREL( RIGHTSON(rqtvertex), ptsroot);
    if
       GETOP(rqtvertex) = AND
    then return SMERGE(leftr, rightr)
    else return PMERGE(leftr, rightr)
    fi
else
    return TREEREL(ptsroot, GETREQTYP(rqtvertex), GETREQMNT(rqtvertex))
fi

end;
```

It is to be noted that Algorithm EXTREEREL is not used directly by the ADVISER program and it is described here in the interest of completeness. The reason for this is that the OVERLORD routine in the program deals with the input compound requirement and has the responsibility for fragmenting it into its constituent atomic requirements. Then partial results for each of the partitions of the graph are generated for these atomic requirements and stored away in anticipation of later repeated use. It is the OVERLORD routine which carries out the combining of the stored partial results. Hence it is only necessary for that routine to call Procedure TREEREL for each PTS for the various atomic requirements in order to pregenerate the partial results and compound requirements never filter down to the PTS package.

5.3 Current Deficiencies in Algorithm TREEREL

In the previous section the algorithm TREEREL was described under the assumption that the root vertex was always necessary for the functioning of the PTS as viewed from the rest of the PMS graph. Thus if some required component was functional at some lower level of the tree it was assumed that all components on the path from that required component to the root vertex would be constrained to work. There are cases where this assumption is not supportable and in this section we examine why.

In Figure 5-7 we show two PTSs in a hypothetical PMS structure. Let us assume that the total overall requirement in computing the reliability of the PMS structure was

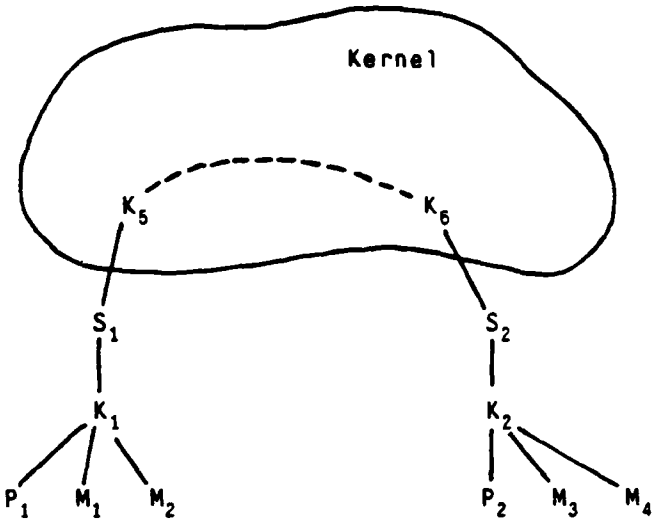


Figure 5-7: Example of TREEREL deficiency.

$$\psi(1,P) \wedge \psi(2,M) \tag{5.4}$$

where the only P and M components in the PMS are those shown in the figure. We see that among other possible success states of the overall PMS structure there is one state in which the components $\{P_1,M_1,M_2\}$ in Figure 5-7 completely satisfy the requirement expression (5.4) which was imposed on the entire PMS structure. When it comes to deciding which components are necessary for these three components to communicate we see that, in the absence of other information, only K_1 is necessary and the components $\{S_1,K_5\}$ are not necessary for communication. However, in this case the TREEREL algorithm will return the following CRP as a partial result for the PTS containing $\{P_1,M_1,M_2\}$:

$$R_{K_5} * R_{S_1} * R_{K_1} * R_{P_1} * R_{M_1} * R_{M_2}$$

This is a pessimistic reliability for this case.

Another possible success state of the PMS in Figure 5-7 under the requirements of expression (5.4) will be that in which $\{P_1,M_3,M_4\}$ are functional. In this case we note that the current TREEREL strategy of requiring all components up to and including the root vertices (K_5 and K_6 in this case) to be functional causes the correct reliability to be computed (assuming that the reliabilities of other necessary components in the Kernel are properly accounted for).

Now assume that the requirement is raised to

$$\psi(1,P) \wedge \psi(3,M).$$

In this instance *all* functional states of the structure will necessarily require that K_5, K_6, S_1, and S_2 be *functional*.

The TREEREL algorithm is seen, therefore, to be deficient currently in cases of requirements on a PMS structure which may be completely satisfied by some *subtree* of a PTS of that structure. The deficiency will make itself evident in such cases and the reliability computed will be pessimistic. The Cm* example in Chapter 7 provides an instance where the deficiency had an effect. Note that there is a possibility of the deficiency arising in the case that the overall PMS structure is itself tree-structured (i.e. no Kernel exists). Appendix A considers the operation of Algorithm GROW under such a circumstance. Some study of the problem will be necessary before a variant of the TREEREL algorithm can be devised which will remove this deficiency.

5.4 Summary

The concept of Pendant Tree Subgraphs (PTSs) was introduced in this chapter. These subgraphs are frequently found in PMS structures in practice and were a natural place to start an investigation into the feasibility of automatic reliability function generation. A procedure was described to recognize symmetric PTSs in a PMS graph. An algorithm was described to compute the reliability of a PTS under a given atomic requirement and a deficiency in this algorithm was noted.

Chapter 6
The OVERLORD routine in ADVISER

6.1 Overview

In this chapter we address the nucleus of the tasks performed in the ADVISER program during the generation of symbolic reliability functions from PMS structures. The Overlord routine constitutes the heart of the reliability evaluator. A diagrammatic representation of the role of the Overlord routine is shown in Figure 6-1. The following broad outline of tasks, and their sequence of occurrence as depicted in the figure, provides a perspective of the entire program against which the Overlord routine is discussed. Following sections will elaborate on these tasks or refer the reader to other chapters where more complete descriptions are available. However, we appeal to the reader's intuition for the duration of this introduction.

In the first phase the input PMS structure is examined to discover physically symmetric substructures within it. These symmetries may then allow some savings in computation at a later phase of the process. Computations are performed with respect to one of a set of symmetric substructures so discovered and then the results of the others in the set are taken to be identical in form. The algorithm SYMMDET for symmetry detection is described in Chapter 4.

The second phase in the process entails the subdivision of the interconnection graph of the input PMS structure into subgraphs in the character of a divide-and-conquer approach. Overall, the vertex set, V, of the interconnection graph G(V,E) is divided into two intersecting subsets, V_{known} and $V_{unknown}$. The set V_{known} represents the set of subgraphs for which specialized techniques are known for calculating symbolic reliability functions. The other vertices, i.e. those in $V_{unknown}$, are treated as a single subgraph, termed the Kernel, and simple path-finding techniques are used for reliability computation in their case since special techniques are unknown. The intersection set $V_{interface} = V_{known} \cap V_{unknown}$ is the set of "interface" vertices between the two sets and these vertices are treated slightly differently from other vertices as their situation demands. At present the ADVISER program has special

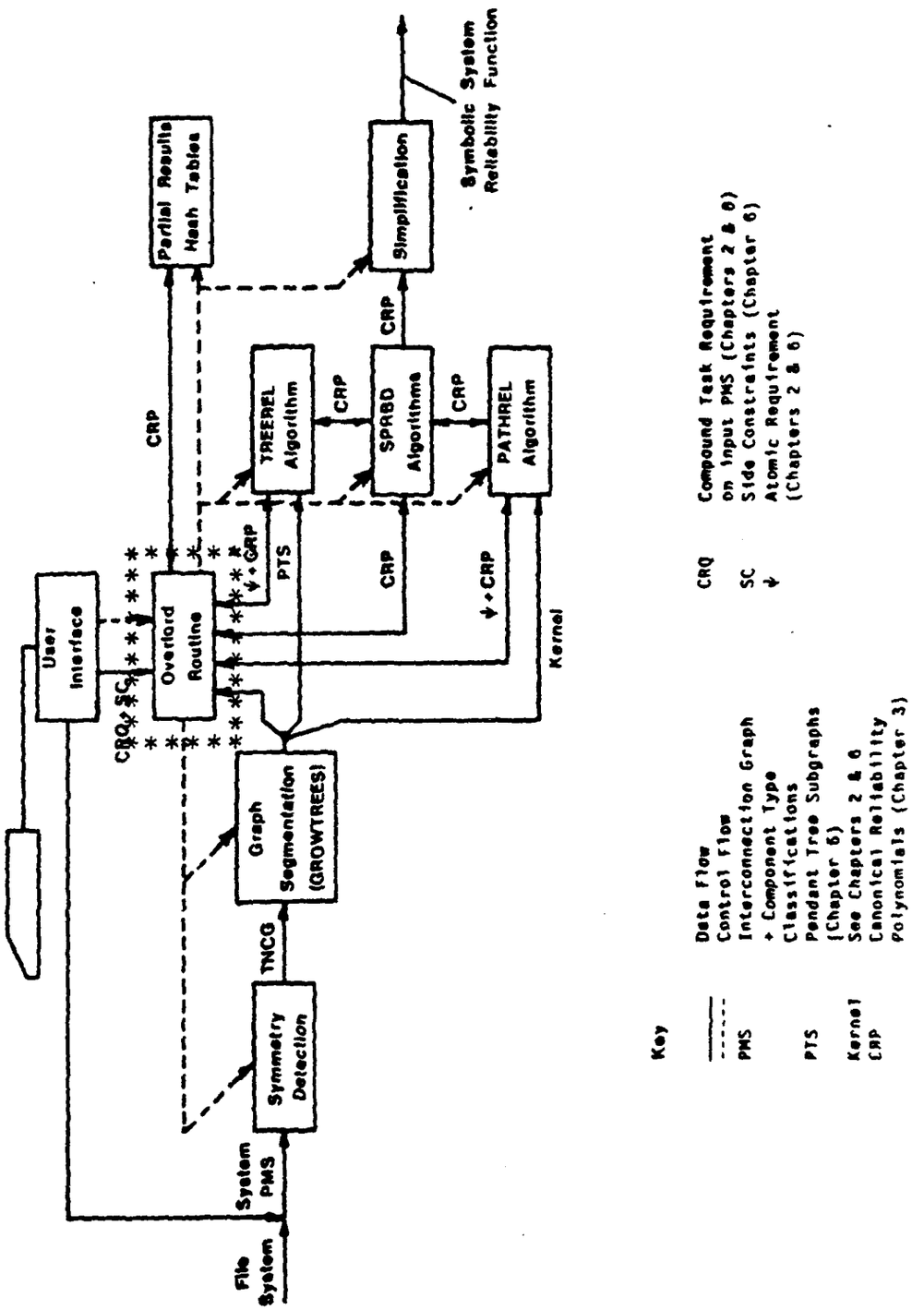


Figure 6-1: The position of the OVERLORD routine in the ADVISER structure.
Also see Page 18.

techniques only for pendant tree graphs and thus the subgraphs represented by V_{known} are all trees. Furthermore, $V_{interface}$ consists of the root vertices of these pendant tree graphs.

In the third phase of the process, the OVERLORD routine operates on one of the program inputs, the compound requirements on the PMS structure which determine its reliability. Its task is then to fragment this compound requirement into various subcases depending on the possibly several ways in which the requirements may be collectively satisfied by the sub-parts of the structure obtained by the subdivision process outlined above. It then accounts for all the various configurations or states of the PMS structure which constitute functional states with respect to those subcases. Clearly, enumeration is involved here but the enumeration is over functional *substructures*, as outlined above, rather than over individual components. *Partial Results*, as we shall refer to them continually throughout this chapter, represent the reliability contribution of parts of the PMS structure under some minimal requirements for functionality. In form they are the Canonical Reliability Polynomials (CRPs) described in Chapter 3. As described below (and briefly in Chapter 2) several partial results regarding each of the substructures are required repeatedly during the enumeration process. The OVERLORD routine anticipates this. It evaluates for each substructure all the partial results that may ever be required in the process thereafter and stores them away in special hash tables for quick access. The evaluation of partial results for a given kind of substructure is done by algorithm(s) which have built-in knowledge of that kind of structure. The low-level package which merges the partial results is called repeatedly during this phase.

In the next phase, the OVERLORD routine uses only the compound requirement and its fragments and the stored partial results. It goes through an enumeration of possibilities in which the substructures collectively satisfy the requirements. During this process the partial results retrieved from the hash tables are merged in one of two possible ways (conjunctive or disjunctive) depending on the structure of the requirements. This is done by calls to the low-level reliability function term list package which operates on CRPs (Chapter 3). At the end of this phase of repeated mergings the reliability function for the PMS structure emerges. For each of the possibilities considered by the OVERLORD routine some subset of the previously generated partial results are used. The generation of possibilities is controlled by the main, and tacit, constraint in the form of the Communication Axiom outlined in Chapter 2. Further pruning of possibilities is done on the basis of side constraints provided by the user of ADVISER as input along with the requirements. The nature of these side constraints is defined by the need to include, in a general though simplified fashion, as much as possible of the semantics typically associated with components in PMS structures.

The system reliability function as generated up to this point is the most general one in which the identity of each individual component in the system is maintained. In other words, assume two identical components c_x and c_y, having identical reliability functions R_x and R_y respectively, ($R_x(t) = R_y(t) = R(t)$), have their reliabilities juxtaposed as factors in some term of the reliability function. Then the product will appear as R_xR_y rather than R^2. The final phase, therefore, carries out the task of simplifying the general reliability function on the basis of the component types which identify each component in the structure as belonging to some generic population of components. It is well to note at this stage that the "simplification" referred to here is limited in nature. It consists of two types of operations, namely

1. the replacing of the product of the symbolic reliabilities of components of like type by the appropriate power of the symbolic reliability of that generic type of component, e.g. replacing R_xR_y by R^2 in the example above, and

2. the algebraic adding of any like terms resulting from operations of the type in Item 1 above, e.g. the set of terms

$$-5R_1R_2R_4^2 + R_1R_2R_4^2 + R_1R_2R_4^2$$

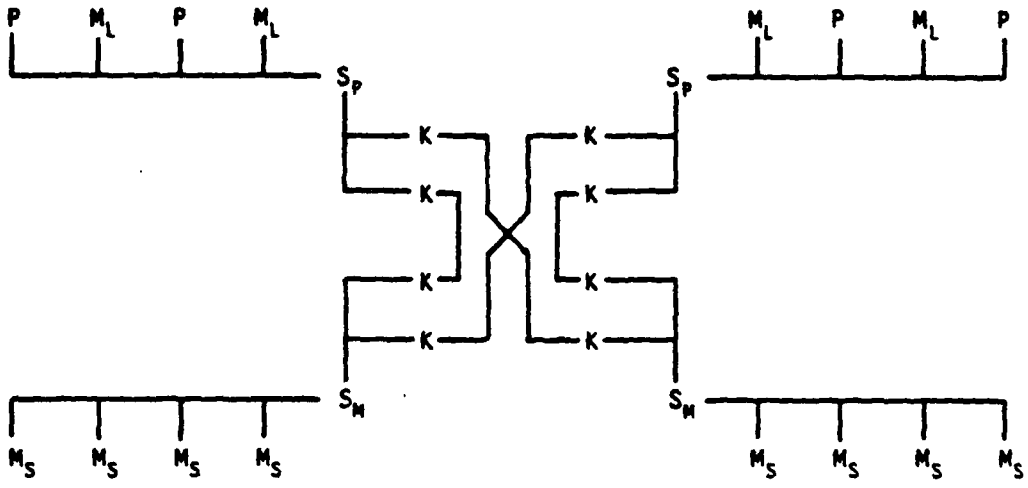
is replaced by $-3R_1R_2R_4^2$.

Thus, for instance, factoring of the final polynomial is not attempted. The techniques for this and other more sophisticated operations were not considered to be the domain of this thesis and the reader is referred to [Macsyma 77] for more information on such techniques.

In following sections we shall consider in turn each of the tasks of the Overlord routine which were outlined in this overview. We shall use the PMS interconnection graph of a Pluribus multiprocessor (Figure 6-2) as a running example at appropriate points in the chapter.

6.2 Detection of physical symmetries in PMS structures

The first task to be performed by ADVISER is the detection of physically symmetric substructures within the given PMS structure. By the physical symmetry of two substructures we imply here not only isomorphism of interconnection graphs of those substructures but also that each pair of corresponding components in the substructures are functionally and statistically identical. We refer to all components coming from the same population as being of the same type, and thus identical in every aspect in which we are concerned during the



Key

P	Processor	M_L	Local Memory
M_S	Secondary Shared Memory	S_P	Processor Bus
S_M	Memory Bus	K	Bus Coupler

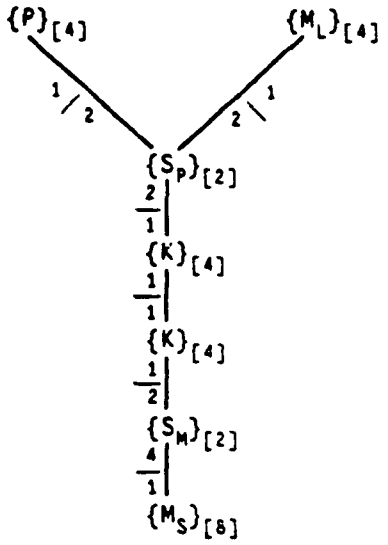
Figure 6-2: PMS structure used as a running example

reliability calculation process.[21] Hence two components of the same type are understood as having identical reliability functions and otherwise being identical. We may then view the PMS interconnection graph as having labelled vertices. The label for each vertex is the type of the component which it represents.

Gaschnig [Gaschnig 77] explored the use of the Neighbors Class Equivalence Relation (NCER) as a basis for determining graph isomorphism. His results were directed toward graphs with homogeneous (i.e. unlabelled) vertices and were used in the study of graphs representing problems in the Artificial Intelligence domain. The Algorithm EDS (Equal-Degree-then-Split), proposed by Gaschnig, partitions the vertex set of a graph of homogeneous vertices into equivalence classes such that two vertices are equivalent iff (1) they have the same degree and (2) multisets consisting of class names of their neighbor vertices are identical. He introduces the notion of the Neighbors Class Graph (NCG) whose

[21] One small operational exception is made in the present implementation. This is described in Section 6.9.1.1.

vertices have a one-to-one correspondence with the equivalence classes resulting from the application of the NCER.



Key

$\{X\}_{[n]}$ ---> equivalence (symmetry) class of components of type X with cardinality n

$\{X\}_{[n]} \overset{p|q}{\text{———}} \{Y\}_{[m]}$ ---> adjacent equivalence classes with connection densities p and q; each X is connected to p Y's and each Y is connected to q X's; np = mq

Figure 6-3: Typed Neighbors Class Graph of the PMS structure in Figure 6-2.

In Chapter 4 we extended the concept of NCER by introducing labelling of each vertex of the PMS graph with the type-name of the component represented by that vertex. We then defined the Typed Neighbor Class Equivalence Relation (TNCER) and introduced the ETEDS (Equal-Type-Equal-Degree-then-Split) algorithm. The latter places two vertices of the PMS graph into the same equivalence class iff (1) their labels are the same, i.e. they represent components of the same type, (2) they are of the same degree and (3) the multisets consisting of the class names of their neighbor vertices are identical. We then defined a Typed Neighbors Class Graph (TNCG) analogous to the NCG of Gaschnig. The inclusion of the additional constraint of equality of component-types as one of the bases for deciding vertex equivalence allows the detection of physically symmetric subgraphs of the PMS graph. Figure 6-3 shows the TNCG for our running example of Figure 6-2.

The application of the ETEDS algorithm to the PMS interconnection graph, G(V,E), produces the TNCG, G′, of G. Intuitively, one may view G′ as the result of "folding" physically symmetric subgraphs of G on top of one another. Thus discovering a subgraph, S′, of some special nature, e.g. a tree, in the TNCG is tantamount to discovering all physically symmetric subgraphs of G that were "folded" to provide S′. Subsequently, if special techniques or closed form solutions are known for the reliability calculation of those kinds of subgraphs, results may be calculated for only one member of the symmetric set and then extended to the others in the set. For instance, k-cliques[22] of G can be deduced from vertices in G′ which have self loops such that the connection densities in both directions on the self loop are equal to k, the cardinality of the class represented by that vertex of G′ (See page 87 in Chapter 4). It is rare to find k-cliques in practical PMS structures but tree subgraphs of a special nature are very often encountered and are thus worthy of study for generation of specific techniques. The special kinds of tree subgraphs of G, referred to here, are what we term Pendant Tree Subgraphs (PTSs). These are tree subgraphs of G such that the one simple path between any one pair of vertices in each PTS is the only such path in G between them. And, in addition, the PTS is separable from G at its root vertex which, therefore, is an articulation vertex of G. This is the only kind of subgraph for which ADVISER currently embodies any special techniques. *However, the structure of the program does not preclude the inclusion of special-techniques for other varieties of subgraphs if and when they are developed.*

6.3 Segmenting the PMS graph; PTSs and the Kernel

Having generated the TNCG G′ of G, the ADVISER program goes on to segment G′ into its PTSs and Kernel. For each of the subgraphs of G′ (and, by implication, of G) generated by this segmentation, symbolic reliability functions will be computed using fragments of the overall compound requirement on G. These constitute partial results which are then merged appropriately to generate the desired symbolic reliability function for G. The motivation for this approach is the anticipated savings in computation time due to the divide-and-conquer paradigm. The segmenting process is seen as the detection of various special types of disjoint subgraphs of G for which there are special techniques known which will generate their symbolic reliability function. Some vertices of G will remain which are not part of any of these special subgraphs. These are then treated by simple path-finding methods, to be described later in this chapter, which will generate the reliability function of the subgraphs of G which

---

[22] A *k-clique* is a complete graph on k nodes. In this instance we are interested in k-cliques which are subgraphs of G.

they collectively represent. This subgraph of "remaining" vertices is termed the Kernel. As noted in the previous section, at this time the special techniques referred to above are known only for the class of PMS interconnection graphs or subgraphs which are Pendant Tree Subgraphs.

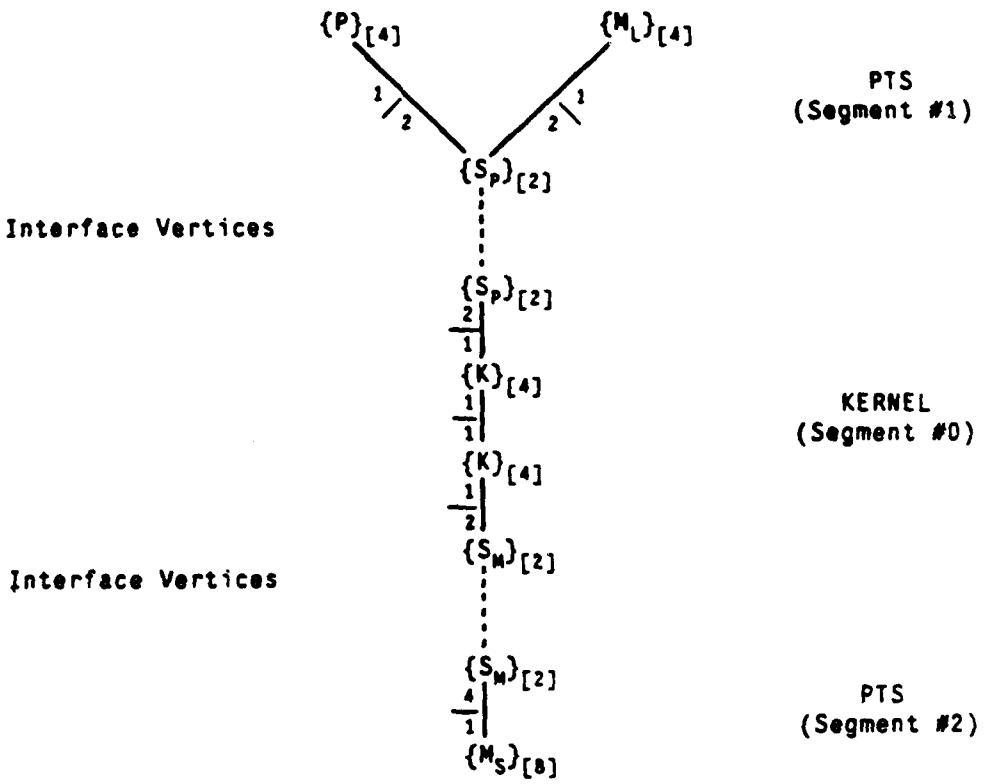
The Pendant Tree Subgraphs of G are generated by Algorithm GROW which was described in Chapter 5. This generation process incrementally adds neighbor vertices to each set of vertices of G' which represent a subtree of some PTS eventually to be generated. Overlapping trees at any stage are then merged into one and the common vertex which caused the overlap becomes part of the root vertex of the larger tree. The termination of the growing process is governed by constraints which determine whether these tree subgraphs of G' represent PTSs of G. After termination of the algorithm, each of the trees thus far grown in G' may represent a set of symmetric PTSs of G. Furthermore, the cardinality of this set of symmetric trees will be the cardinality of the equivalence class which is represented by the root vertex of the tree in G'.

The discovery of the symmetric PTSs of G establishes a basis for the subdivision of G into PTSs and the Kernel. This subdivision or segmenting is shown diagrammatically in Figure 6-4 for our example PMS interconnection graph.

At this stage of the computation the ADVISER program builds a very important table, the Segments Table, which is the repository of information regarding the nature of the segments of G. For the purposes of this table it is sufficient to store information regarding all of the segments except for the Kernel. Information about the latter can be deduced from the former. Thus, for instance, to find out what vertices in the vertex set V of G fall into the Kernel, it is sufficient to subtract from V the union of the vertex sets of all the other segments. *Henceforth we shall use the term "segment" to refer generically to the subdivisions of the graph G. Those segments for which special techniques are known will be termed "known-segments" and the remaining vertices and arcs of G will be collectively termed the "Kernel".*

In the current version of the program, all the known-segments are tree graphs since special techniques have been developed only for these. Each known-segment as it is generated is assigned a unique integer in the sequence starting from zero. Then information regarding the known-segment is stored at the location in the Segments Table indexed by this integer. The following major items (in addition to bookkeeping information) comprise that information:

- The vertex set of the known-segment



Key

PTS Pendant Tree Subgraph

Segment #n Arbitrary assigned indices into Segments Table

Figure 6-4: Segmentation of PMS structure of Figure 6-2 into Pendant Tree Subgraphs and the Kernel

- The set of indices of the equivalence (symmetry) classes, induced by the symmetry detection algorithm of Chapter 4, such that the union of the classes indexed by this set is the vertex set of the known-segment in G.

- The index of the symmetry class of vertices in this known-segment which are the interface vertices at which the known-segment is connected to the Kernel. In general this should be a *set* of symmetry class indices. However, since known-segments currently are tree structures only, each interface vertex symmetry class will have only one member; the root of the tree.

The Segments Table along with the data structures which store the original graph G and the equivalence classes on G in the form of the TNCG G', completely characterize the work done

by the program upto this phase. In addition, each known-segment, and information about it, is accessible by means of a unique index into the Segments Table and, therefore, is completely characterized by that index. *This complete and unique characterization of a known-segment by its Segments Table index becomes very important in later phases (see Section 6.5.1.1)*

6.4 Requirements on the PMS structure

In this section we shall discuss how the Disjunctive, Conjunctive and Atomic requirements (see Chapter 2) are employed by ADVISER in producing the system reliability function. The material in this section is complementary to a similar discussion in Chapter 2. It is also prefatory in nature to the subsequent sections in this chapter. These latter sections deal with the generation and combining of partial results on the way to achieving the final goal of a symbolic reliability function for the PMS structure. The paradigm of fragmenting and distributing these requirements into simple cases over the various segments of the interconnection graph G is illustrated below. This process effects the enumeration of the gross cases of system functionality while lower levels of detail are subsumed within the operation of the special-technique algorithms. These algorithms provide the partial results by operating on the known-segments of G for the very simple cases of requirements generated by the fragmenting of the overall requirements. The partial results are then recombined to produced the desired final result. *This paradigm is used throughout the Overlord process and is independent of the nature of the known-segments so long as special-technique functions exist for those segments and are callable by Overlord.*

A typical conjunctive requirement on our example PMS structure of Figure 6-2 could read:

$$\psi(1,M_S) \wedge \psi(1,P) \wedge \psi(1,M_L)$$

Likewise, a typical disjunctive requirement might read:

$$\psi(1,P) \wedge (\psi(1,M_L) \wedge \psi(2,M_S) \vee \psi(2,M_L) \wedge \psi(1,M_S))$$

Each of these requirement expressions states the conditions under which the system in the example is considered functional for some task. Note that these requirement expressions are abbreviated statements of those conditions. In other words, not all components, which need to function in order to insure system success, are referred to in these expressions. One such unmentioned component is an S_P in Figure 6-2 which needs to function if its P needs to store information into its M_L. Thus a distinction arises between Critical Components (such as M_L and P by virtue of being referenced in the requirements above) and Auxiliary Components

(such as S_p). *The ADVISER program assumes that all component types referred to in the atoms of the input compound requirement are critical component types.*

6.4.1 Atomic Requirements

Let us first consider an atomic requirement $\psi(N,t)$, $N>0$, and observe how it may be satisfied by a PMS interconnection structure G to provide functional system states. Assume, for the sake of argument, that the PMS structure under analysis has been segmented into m distinct segments. Furthermore, assume that in each of these m segments of G there are present $a_i \geq N$ components of type t. Figure 6-5 shows the various ways in which the required N components of type t may be chosen from the m partitions in order to satisfy the atomic requirement. This procedure is analogous to assigning N balls to m distinct ordered urns so that any urn contains zero or more balls when each urn has the capacity to hold at least N balls. In other words, it is possible to assign all N balls to a single urn. There are

$$\binom{N+m-1}{N}$$

ways of doing this[23] and these are shown in Figure 6-5. Each m-dimensional vector $P_N = (p_1,p_2,\ldots,p_m)$ such that

$$\Sigma_{i=1}^{m} p_i = N \quad p_i \geq 0, \text{ integral} \tag{6.1}$$

is called an m-composition of the integer N.

We next consider the possibility, which is indeed most probable, that some of the m segments may contain less than N components of type t and some segments may contain none at all of type t. The effect of this restriction is to place upper bounds on the value of integers in particular columns of Figure 6-5. Then each m-composition of N which does not meet these upper bounds is removed from further consideration since there can never be a functional system state which is composed of that particular distribution of components of type t amongst the m segments.

Definition 6.1: Assume that the segmenting process of earlier sections leaves the PMS interconnection graph divided into $m \geq 1$ distinct segments. A Capacity Vector of G with respect to some component type t is defined as

$$\chi_t = (c_{1t},c_{2t},\ldots,c_{mt})$$

---

[23] see Nijenhuis and Wilf [Nijenhuis 78] for a lucid explanation of this result.

Choices	Segments p_1	p_2	p_3	p_4	.....	p_m
1	N	0	0	0	.....	0
2	N-1	1	0	0	.....	0
3	N-1	0	1	0	.....	0
4	N-1	0	0	1	.....	0
...	...	...	...	...	.....	...
...	...	...	...	...	.....	...
m	N-1	0	0	0	.....	1
m + 1	N-2	2	0	0	.....	0
m + 2	N-2	1	1	0	.....	0
m + 3	N-2	1	0	1	.....	0
...	...	...	...	...	.....	...
...	...	...	...	...	.....	...
2m-1	N-2	1	0	0	.....	1
2m	N-2	0	2	0	.....	0
2m + 1	N-2	0	1	1	.....	0
...	...	...	...	...	.....	...
...	...	...	...	...	.....	...
...	N-2	0	0	0	.....	2
...	N-3	3	0	0	.....	0
...	N-3	2	1	0	.....	0
...	N-3	2	0	1	.....	0
...	...	...	...	...	.....	...
...	...	...	...	...	.....	...
...	N-3	2	0	0	.....	1
...	N-3	1	2	0	.....	0
...	N-3	1	1	1	.....	0
...	...	...	...	...	.....	...
...	...	...	...	...	.....	...
...	...	...	...	...	.....	...
...	...	...	...	...	.....	...
$\binom{N+m-1}{N}$	0	0	0	0	.....	N

Figure 6-5: Choosing N components from m segments; m-compositions of the integer N.

where segment i of G contains c_{it} components of type t and c_{it} is the Capacity of segment i with respect to component type t.

Type Number	Type Name	Segment Number 0	1	2
0	P	0	4	0
1	M_L	0	4	0
2	S_P	0	2	0
3	S_M	0	0	2
4	M_S	0	0	8
5	K	4	0	0

Note: Interface vertices (e.g. S_P,S_M) are counted in the PTS segments for satisfying requirements but are used in the Kernel for generating path reliabilities (see Section 6.7.2).

Figure 6-6: Capacity vectors for the PMS of Figure 6-2 when segmented as in Figure 6-4.

Figure 6-6 shows the capacity vectors for the example PMS structure of Figure 6-2. On the basis of the upper bounds, or capacity, of the segments we may divide the $\binom{N+m-1}{N}$ possible m-compositions of N into two groups, namely Feasible Compositions and Infeasible Compositions.

Definition 6.2: An m-composition P_N of the integer requirement N of an atomic requirement $\psi(N,t)$ over the m segments of G is said to be feasible iff[24]

$$P_N \leq \chi_t$$

An m-composition P_N under the above conditions is said to be infeasible iff P_N is not feasible.

[24] If $A=(a_1,a_2,\ldots,a_m)$ and $B=(b_1,b_2,\ldots,b_m)$ are two m-dimensional vectors then we shall say that $A \leq B$ iff $\forall i = 1,\ldots,m,\ a_i \leq b_i$

Reverting to our urn model we find this above situation analogous to one in which the i^{th} of the m urns has a maximum capacity $0 \leq c_i \leq N$. Then some of the ways of distributing N balls among the urns as depicted in Figure 6-5 are impossible. These impossible distributions of balls in urns correspond to infeasible compositions. The possible distributions correspond to feasible compositions.

For each feasible composition, $P_N = (p_1, p_2, \ldots, p_m)$, the requirement $\psi'(p_i,t)$ is applied to segment i and a partial result is generated for that segment under that requirement. We shall term $\psi'(p_i,t)$ a Fragment Requirement of $\psi(N,t)$. Whatever special techniques are applicable to the segment are applied at this point. For instance, Algorithm TREEREL is applied to segments which are PTSs. Of course, if $p_i = 0$ then segment i will not participate in the satisfaction of this particular P_N. The partial results for the individual segments under the $\psi'(p_i,t)$ are then SMERGEd[25] to generate a partial result for this particular feasible P_N.

The process is repeated, and a partial result generated, for each feasible m-composition of N in $\psi(N,t)$. Now, any one of the feasible compositions applied to the m segments of G satisfy the atomic requirement $\psi(N,t)$. Therefore, finally, the partial results for all the feasible m-compositions are PMERGEd.[26] The result of this process is then a symbolic reliability function (in internal term-list form) for G under the atomic requirement $\psi(N,t)$.

6.4.2 Compound Requirements

6.4.2.1 Conjunctive Requirements

We now broaden our scope to include compound requirements. To begin we shall consider conjunctive requirements and later extend the conclusions to disjunctive requirements. The former consist of atoms operated on solely by the conjunction operator AND. They imply that for system success all the individual atomic requirements must be satisfied in conjunction by G. We shall represent a conjunctive requirement as

$$\bigwedge_{i=1}^{K} \psi_i(N_i,t_i)$$

---

[25] The SMERGE algorithm is described in Chapter 3 and it may be briefly characterized as computing the probability of a conjunction of events. In this instance all the $\psi'(p_i,t)$ must be satisfied in conjunction by the segments of G in order for G to satisfy $\psi(N,t)$.

[26] The PMERGE algorithm is described in Chapter 3 may be characterized as forming the probability of a disjunction of events.

where the ψ_j are atomic requirements of the type "at least N_j of component type t_j." Each atom $\psi_j(N_j,t_j)$, $j = 1,...,K>0$, in the conjunctive requirement is considered to refer to a different critical component type t_j.[27]

Again, returning to our ball and urn analogy we now have the same m urns (partition of G) but K sets of colored balls with balls in set j being of color t_j. Again we initially assume that each urn has an unlimited capacity for balls. We see that the different ways of assigning $N_1 + N_2 + ... + N_K$ balls to m urns so that each urn has zero or more balls of each color is

$$\prod_{j=1}^{K} \binom{N_j + m - 1}{N_j}$$

This is the total number of ways of minimally satisfying the conjunctive requirement in G. It assumes, of course, that each segment of G is individually capable of minimally satisfying each of the atomic requirements of the conjunctive requirement. Again, we note that, in general, this last statement will not be true and that for each atomic requirement, $\psi_j(N_j,t_j)$, there will be feasible and infeasible m-compositions of N_j.

The paradigm used by the ADVISER program in the case of conjunctive requirements is to first compute the capacity vectors for G with respect to all the critical component types (i.e. the component types $t_1,...,t_K$ referred to in the conjunctive requirement). Then, all the feasible m-compositions of each N_j in $\psi_j(N_j,t_j)$, $j = 1,...,K$, are generated sequentially. *However, the next feasible m-composition of any N_j is generated only after cycling through all possible feasible m-compositions of N_{j+1}.* Thus, all possible combinations of all feasible m-compositions of each of the N_j are produced in sequence. For each such combination, a partial result is computed in the manner of Section 6.4.1 for each m-composition within that combination. These partial results are then SMERGEd to provide a partial result representing that *combination* of m-compositions. The SMERGEing is indicated due to the conjunctive nature of the requirement. The process of SMERGEing all the partial results for the atomic requirements in a combination of m-compositions is carried out for all the possible combinations of *feasible* m-compositions. Finally, since any one of the combinations represents a possible way of satisfying the conjunctive requirement, the partial results for combinations are then PMERGEd. What results from this final PMERGE operation is the reliability function for G under the conjunctive requirement

---

[27] If this is not true, i.e. for some i and j $t_i = t_j$, then due to the concept of conjunction we may, without affecting the outcome, discard ψ_i and retain ψ_j if $N_j > N_i$ and *vice versa*

$$\bigwedge_{j=1}^{K} \psi_j(N_j,t_j)$$

Figure 6-7 shows the sequence in which combinations of feasible m-compositions are generated for an example case where G is subdivided into m = 3 segments and there are K = 3 atoms in the conjunctive requirement. Each group of three columns shows the sequence of m-compositions for one of the atomic requirements. Each column, in each group of three, represents one of the three segments of G. At the head of the j^{th} group of three columns is the capacity vector for G with respect to the critical component type t_j, j = 1,2,3. Thus, for instance, in the case of t_3 (third group of columns), segment # 1 contains three components of type t_3. Likewise, segments # 2 and # 3 contain one and two components of type t_3 respectively. In the case of critical component type t_2, segment # 2 contains no components of this type and thus no feasible 3-composition of $N_2 = 1$ will have an integer greater than zero in this column. The partial result for a combination in some row of the figure is obtained by the SMERGE operation on the partial results obtained by applying the atomic requirements within that row to the appropriate segment (see Figure 6-7 for the case of row 7). The final result is obtained by the PMERGE function on the partial results for the combinations.

6.4.2.2 Disjunctive Requirements

The sequential generation of combinations of feasible m-compositions as depicted in Figure 6-7 is inadequate in the case of disjunctive requirements. This is because not all of the atomic requirements of a disjunctive requirement may need to be satisfied simultaneously. Take, for example, the following disjunctive requirement:

$$\psi_d = \psi_1(N_1,t_1) \wedge \psi_2(N_2,t_2) \vee \psi_3(N_3,t_3)$$

Here, for G to satisfy ψ_d, there are three possible cases, namely

1. satisfy ψ_1 and ψ_2 simultaneously but not ψ_3,
2. satisfy ψ_3 alone, and
3. satisfy ψ_1, ψ_2 and ψ_3 simultaneously

However, the process of Figure 6-7 admits of the third case alone, thereby missing valid possibilities. The tack taken to solving this problem in ADVISER is to convert every disjunctive requirement received as input into a "sum-of-products" canonical form. In other words a conversion is made to a disjunctive normal form. Then for each of the purely conjunctive requirements in this canonical form, the process of section 6.4.2.1 is followed to generate a partial result *for the conjunctive requirement*. Finally, the partial results for the conjunctive

Atomic Requirement →	$\psi(N_1,t_1),N_1=2$			$\psi(N_2,t_2),N_2=1$			$\psi(N_3,t_3),N_3=1$			Partial Results for Combinations of 3-Compositions
Segment Index →	#1	#2	#3	#1	#2	#3	#1	#2	#3	
Capacity Vectors →	1	0	2	1	0	1	3	1	2	
	(N_{11})	(N_{12})	(N_{13})	(N_{21})	(N_{22})	(N_{23})	(N_{31})	(N_{32})	(N_{33})	
	1	0	1	1	0	0	1	0	0	R_1
	1	0	1	1	0	0	0	1	0	R_2
	1	0	1	1	0	0	0	0	1	R_3
	1	0	1	0	0	1	1	0	0	R_4
	1	0	1	0	0	1	0	1	0	R_5
	1	0	1	0	0	1	0	0	1	R_6
	0	0	2	1	0	0	1	0	0	R_7
	0	0	2	1	0	0	0	1	0	R_8
	0	0	2	1	0	0	0	0	1	R_9
	0	0	2	0	0	1	1	0	0	R_{10}
	0	0	2	0	0	1	0	1	0	R_{11}
	0	0	2	0	0	1	0	0	1	R_{12}
										R_{FINAL}

LEGEND:

Each $\psi_j(N_j,t_j)$ is fragmented into $\psi'(N_{j1},t_{j1})$, $\psi'(N_{j2},t_{j2})$, and $\psi'(N_{j3},t_{j3})$, where $N_{j1}+N_{j2}+N_{j3}=N_j$ and N_{j1}, N_{j2}, N_{j3} are the values in each row of the j^{th} 3-column group in the table above.

Let *r(a,b,c)* be the Partial Result CRP for segment *a* under the atomic requirement $\psi(b,c)$, obtained by algorithms such as TREEREL. Then, for example, R_7 above is given by

$$R_7 = r(3,2,t_1) \circledast r(1,1,t_2) \circledast r(1,1,t_3)$$

and

$$R_{FINAL} = R_1 \circledast R_2 \circledast R_3 \circledast \ldots \circledast R_{12}$$

R_{FINAL} is the CRP under the conjunctive requirement $\psi_1(2,t_1) \wedge \psi_2(1,t_2) \wedge \psi_3(1,t_3)$

Figure 6-7: Derivation of partial and final results for a conjunctive requirement

requirements within the canonical form are operated upon by PMERGE to obtain the final result for the original disjunctive requirement.

Hence, for example, the following disjunctive requirement

$$(\psi_1 \vee \psi_2) \wedge (\psi_3 \vee \psi_4)$$

is converted to the canonical form

$$(\psi_1 \wedge \psi_3) \vee (\psi_1 \wedge \psi_4) \vee (\psi_2 \wedge \psi_3) \vee (\psi_2 \wedge \psi_4)$$

which is a *pure disjunction of conjunctions*. Then the individual conjunctive requirements, e.g. $(\psi_1 \wedge \psi_4)$ are handled as described in Section 6.4.2.1.

6.4.3 Efficiencies in the handling of requirements

To conclude the discussion on requirements we shall consider two types of efficiencies introduced in the design of the ADVISER program in the handling of requirements. They have to do, respectively, with the *a priori* generation and storage of partial results for repeated later use and the deferment of the combining of these partial results to a final phase of the program where some savings in computation are possible.

There is a third type of efficiency introduced into ADVISER which only tangentially impinges on the issue of requirements. It has more to do with the use of symmetries discovered in the PMS interconnection graph and, as such, it is addressed in a subsequent section.

6.4.3.1 Pre-generation of partial results

It will be noted from Figure 6-7 that the 3-compositions all occur repeatedly during the enumeration process. This in turn manifests itself as the repetition of integers in any one of the columns. Now, the occurrence of an integer, say k, in a column represents the application of a fragment atomic requirement, $\psi'(k,t)$, on the segment of G represented by the column. The critical component type t is the same one referred to in the atomic requirement of which $\psi'(k,t)$ is a fragment. In a simple-minded version of the OVERLORD procedure, the algorithms to generate the partial results under $\psi'(k,t)$, for the given segmentation of G, would be called repeatedly to regenerate the same partial result each time. However, it is feasible to compute this partial result once and store it away in a table, thereby avoiding such repeated calls. The ADVISER program does precisely this.

Precomputation raises the question of predicting what partial results would ever be needed in the course of computing the reliability function. A little thought shows that a simple solution is as follows. For a given segment i and a critical component type t, say the capacity of the segment is c_{it}. Recall, also, that a component type t is labelled "critical" if it appears in some atom $\psi(N,t)$ of the input compound requirement. If $c_{it} = 0$ then $\psi'(N,t)$ is not applicable to segment i and the question does not arise. If $c_{it} > 0$ then the fragments of $\psi(N,t)$, which could possibly be fruitfully applied to segment i, are

$$\psi'(1,t), \psi'(2,t), \ldots, \psi'(\mu,t)$$

where

$$\mu = \begin{cases} N_{max} & c_{it} > N_{max} \\ c_{it} & c_{it} \leq N_{max} \end{cases} \quad \text{(see below)}$$

The appearance of N_{max} is explained by the fact that, in general, for some critical component type t, there may be several atoms in the compound requirement which refer to t (this is true of a disjunctive requirement). If these atoms are

$$\psi(N_1,t), \psi(N_2,t), \ldots, \psi(N_r,t)$$

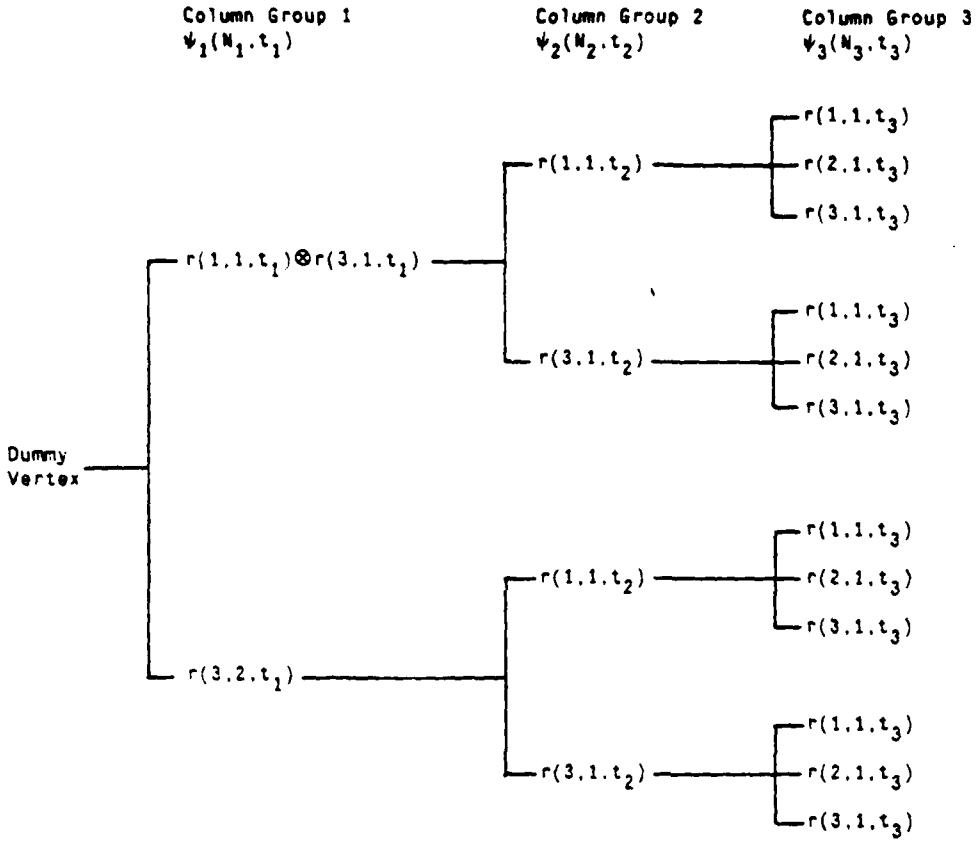
then

$$N_{max} = MAX(N_1, N_2, \ldots, N_r)$$

It is necessary to use N_{max} since we are concerned to develop <u>all</u> possible fragment requirements which may ever be applied to segment i during the computation of the system reliability function.

6.4.3.2 *Deferring the combining of partial results*

An examination of Figure 6-7 shows that (using the nomenclature of the figure) the partial results R_1 through R_6 all share the same fragments of $\psi_1(2,t_1)$ in the first group of three columns. Likewise, R_7 through R_{12}. Furthermore, R_1 through R_3 additionally share the same fragments of $\psi_2(2,t_2)$ in the <u>second</u> group of three columns in the figure. Likewise, R_4 through R_6, R_7 through R_9 and R_{10} through R_{12}. The figure suggests that each of the R's are computed separately and then finally PMERGEd to obtain R_{FINAL}. A more efficient way to accomplish this, however, is to make use of the fact that the PMERGE and SMERGE operations are distributive over each other. We may "factor" out common partial results and merge them only at the appropriate time so that repetitious mergings of the same partial result are avoided wherever possible. The information in Figure 6-7 may be represented equivalently



LEGEND:

r(a,b,c) is the Partial Result CRP for segment a under the atomic requirement ψ(b,c). This figure is to be read in conjunction with Figure 6-7

Figure 6-8: CRPTree for the example of Figure 6-7

by the tree in Figure 6-8. Each level in the tree corresponds to a 3-column group in Figure 6-7. However, partial results are displayed instead of the requirement fragments. Thus information in a column group of Figure 6-7, which is common to several rows in the column group to its right, is condensed into a single tree node. Since there are no columns to the right of the last column, the tree will always have as many leaves as there are rows. We shall term this a Canonical Reliability Polynomial Tree (CRPTree) since the labels of its vertices are Canonical Reliability Polynomials which were treated in Chapter 3. R_{FINAL} may be calculated from it using the following algorithm by calling the procedure CRPTREEMERGE on the root of the CRPTree.

Algorithm CRPTREEMERGE

This effects the merging of partial results (Canonical Reliability Polynomials) stored in the CRPTree.

Notation:

- PR(v) is a function which returns the partial result stored in vertex v of the CRPTree.

- CRP is an abstract data type which holds a canonical reliability polynomial as a value.

```
Procedure CRPTREEMERGE (treevertex)
begin CRP aggregate;
  aggregate ← NIL   ! recall that PMERGE and SMERGE of a null CRP
                    with a non-null CRP A, returns A itself
  foreach son in successors of treevertex
  do  aggregate ← PMERGE( aggregate, CRPTREEMERGE(son) );
  return SMERGE( aggregate, PR(treevertex) )
end;
```

It may be noted in passing that the odometer analogy to the generation of combinations of m-compositions mentioned earlier, is reflected in the CRPTree. Each level of the tree corresponds to a wheel of the odometer with each vertex at that level holding one of the values which appear on the wheel. Hence, the tree may be generated during the process of generation of the combinations of m-compositions. The ADVISER program uses this strategy and postpones most of the merging of the partial results in this fashion to a final phase of the processing of any conjunctive requirements. The CRPTree, its construction and its use are described more fully in Section 6.8.1.

Experience with ADVISER has shown that the largest part of the computation time used by the program is spent in the phase where the partial results in the CRPTree are merged to produce R_{FINAL} for a conjunctive requirement. The simple algorithm CRPTREEMERGE given above, though correct, is not nearly as efficient as could be desired. This issue will be addressed in a subsequent section.

6.5 Generation of Partial Results for PTSs

Section 6.4.3.1 demonstrated that it was possible, *a priori*, to compute and store away all the partial results which might be needed in later program phases. In this section we shall examine this process of prior generation of partial results for PTSs, how these partial results are hash coded away for later access, and what savings in computation are afforded by existence of structurally symmetric PTSs in the system.

6.5.1 Symmetric PTSs

It will be recalled that each segment entered in the Segments Table is a PTS of G'. Hence each known-segment can possibly represent several symmetric PTSs of G which were equivalenced as a result of the NCER algorithm. Indeed, the cardinality of the equivalence class represented by the root vertex of the known-segment is the number of such symmetric PTSs of G (i.e. images of the PTS in G') which were equivalenced. *It is emphasized here that the underlying model supports known-segments which are arbitrary subgraphs of G'. The only requirement is that special techniques be available for the reliability computation of the images in G of those types of subgraphs.* Each such subgraph of G' would then potentially represent a set of symmetric images in G. Since these images are not necessarily tree graphs they will, in general, have more than one interface vertex by which they are connected to the Kernel. Then an analysis of the equivalence classes which hold the interface vertices of these images, using Theorem 4.8 and related results of Chapter 4, will reveal the number of such symmetric images in G of the segment of G'.

Returning to the current version of ADVISER in which the known-segments of G' will always be PTSs of G', there are, therefore, two possibilities, namely: (1) a known-segment represents a unique PTS of G and (2) a known-segment represents several symmetric PTSs of G. The former case presents no problem and is handled directly in the manner of Section 5.2.1 using the TREEREL algorithm. In the latter case there now exists the possibility of economizing on computation. By our definition of physical symmetry in Chapter 4 two PTSs of G are symmetric iff their interconnection graphs are isomorphic and corresponding vertices in the two PTSs represent identical components. Hence any reliability function derived under a given requirement for one of a set of symmetric PTSs, will be an exemplar or template for the reliability function under the same requirement for any member of the set.

It is evident that the sets of components represented by the symmetric images in G of a PTS of G' are disjoint. Two conclusions of this are to be strongly emphasized here, namely

1. By the fundamental assumption of statistical independence of component failure behavior, the reliability functions derived for two symmetric images represent events which are statistically mutually independent. Although the two images are symmetric they have two physically disjoint sets of components

2. Assume that two reliability functions are derived for the same image in G for two different requirements. These functions provide the probability of two events which may be mutually dependent since they may depend on intersecting sets of components within the image. Hence it must be remembered that these reliabilities cannot simply be multiplied in algebraic manipulations.

These two conclusions have strong ramifications later during the operation of the SPRBD algorithm wherein the form of the result of the SMERGE operation on two canonical reliability polynomials is dependent on whether those polynomials represent the probabilities of events which are statistically dependent or statistically independent.

6.5.1.1 Unique identification of PTS partial results

The possibility of a given PTS of G′ representing a set of two or more symmetric PTSs of G raises the question of uniquely identifying each member of such a set. Note that within a set of symmetric PTSs of G the unique identities, in G, of their root vertices are sufficient for such a purpose. However, further information is required in ADVISER in the interest of efficiency. Most of this information is precomputed once and stored in the Segments Table. Hence, any given PTS of G is identified completely and uniquely by the following two items of information:

1. The index of its root vertex.

2. The index of the Segments Table entry which describes the known-segment of G′ which represents it.

6.5.1.2 The Templates Table

We noted above that it was sufficient to compute a partial result for a given atomic requirement for any one of a set of symmetric PTSs in G. Since partial results for a set of symmetric PTSs under the same requirement have the same form, the template may be stored instead of the individual CRPs. This is done using a special table termed the Templates Table. All unique templates ever generated during a run of ADVISER are assigned unique entries in this table. Unique and complete specification of any partial result is achieved with the following five items of information about it:

1. The unique index of the root vertex of the PTS for which this partial result was derived.

2. The index of the Segments Table at which resides information about the known-segment which represents the PTS in Item 1

3. The unique index of the component type specified in the atomic requirement for which this partial result was computed (i.e. the y in $\psi(x,y)$).

4. The number of components of the given type in Item 3 which are required to be functional (i.e. the x in $\psi(x,y)$).

5. The index in the Templates Table at which the template for this partial result is stored. This index is, however, obtained going indirectly through the Factors Table, the purpose of which is explained below.

The Items 1 and 2 identify a partial result as referring to a unique PTS of G within a known-segment of G'. The Items 3 and 4 further identify the partial result as having been derived for a given $\psi(x,y)$. The last Item, 5 implements the space-saving device by pointing to the appropriate template.

A partial result is thus uniquely specified by a five-tuple of integers corresponding to the five items above. This five-tuple is also the key used during the hashing and retrieval process. The interrelations between partial results, PTSs, the Segments Table and the Templates Table are depicted diagrammatically in Figure 6-9.

As was described in Section 6.4.3.1, partial results may be used repeatedly during the construction of the CRPTree; indeed they usually are due to the combinatorial backtrack nature of generating all possible compositions of requirement integers over all segments. The CRPTree is eventually collapsed by merging the partial results stored in it at each node, from the leaves up to the root and breadth-first at each level. However, in doing so, quite often two copies of the same partial result could be SMERGEd or PMERGEd. Now, due to the idempotence of the SMERGE and PMERGE operations, attempting to merge identical partial results constitutes a waste of compute time. This problem is the subject of the following subsection.

6.5.1.3 The Factors Table

A partial solution to the problem of multiple use of the same partial result arises from the observation that each partial result can be uniquely characterized. This allows each partial result to be assigned a unique bit position in the AUXVEC bitvector of CRP terms (see Chapter 3). The partial result can then be assigned a literal and becomes a "factor" of the CRP term. The association between such factor-polynomials and their assigned bit position is made in the Factors Table. For ease of implementation the bit position of a particular factor-polynomial

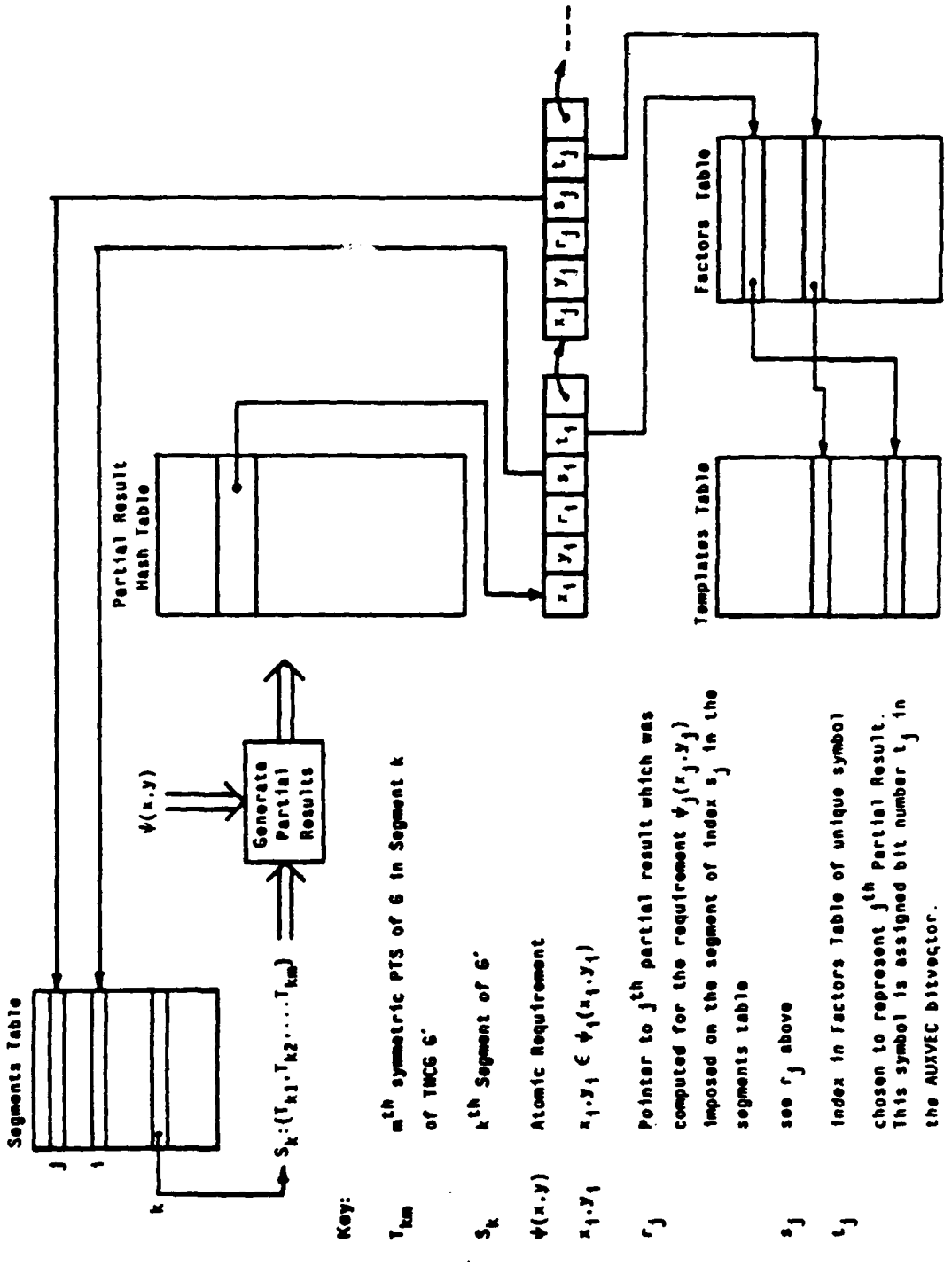


Figure 6-9: The relationship of important tables in ADVISER

is simply the index of its entry in the Factors Table. The reader is again referred to Figure 6-9. Note that the arrangement allows factor polynomials to share the same template.

6.6 The Communication Axiom and the Kernel

In Chapter 2 we introduced the Communication Axiom which is the basis of the reliability calculation paradigm used by the ADVISER program. We shall now restate the Axiom and some associated definitions in order to clarify the discussion in regard to the computation of partial results for the Kernel. The reader is referred to Chapter 2 for introductory remarks and background context.

6.6.1 The Communication Axiom

In Sections 6.4.1 and 6.4.2.1 we discussed feasible and infeasible compositions of the requirement integers and the computation of partial results for combinations of feasible compositions. However, as was pointed out, a combination of feasible compositions is only a necessary prerequisite to a case where the system is functional. In addition, the combination of feasible compositions must be such that the Communication Axiom is satisfied in order for the system to be functional under that particular choice of critical components. In other words, in addition to having a functional minimal critical resource set, the critical components in that set must also have functional pathways in the structure in order to communicate information amongst themselves. We state these ideas more formally in the following:

Let $T = \{t_i\}$, $i \in \{1,2,...,n\}$, be the set of component types specified in the minimal requirements input. The set T is then, by default, the set of critical component types.

Let $Q_i = \{q_{ij}\}$, $j \in \{1,2,...,m_i\}$ be the set of all identical components of type t_i present in the structure.

Let $T' \subseteq T$ and let $M_k \subseteq Q_k$, $t_k \in T'$.

If $M' = \bigcup_k M_k$ is a set of critical components such that the boolean statement of minimal requirements is satisfied minimally then M' is a Minimal Critical Resource Set (MCRS).

A simple path p_{ab} between any two vertices v_a and v_b in G is said to be a functional path iff all components represented by vertices along that path are functional.

Let V_M be the set of vertices in G that represent the components in M′, an MCRS of G.

Definition 6.3: We define a Communicability Graph or K-Graph, $G_K(V_K,E_K)$, for M′ as follows:

- There is a bijective mapping from the vertex set V_K to the vertex set $V_{M'}$.
- A Communicability Edge or K-edge, $(v'_K,v''_K) \in E_K$ will exist iff at least one functional path exists between the vertices in G which are the images, under the bijective mapping, of v'_K and v''_K in G_K respectively.

Axiom 6.1: Communication Axiom: For any MCRS, M′, of the system represented by G, if the components in M′ are all functional, then the system will be functional iff the K-graph of M′ is connected.

The Communication Axiom is used during the computation of partial results for the Kernel which is the subject of attention in the following subsections.

6.6.2 The Kernel

As was described in Chapter 2, a symmetry detection process enables the identification of segments of the PMS graph, G(V,E), for which special techniques are available for reliability computation. These known-segments are treated separately, in isolation from the rest of G. Each such known-segment is attached to the rest of G via a set of one or more interface vertices. At present pendant tree subgraphs (PTSs) are the only such known-segments treated and, therefore, each will have only one interface vertex connecting it to the rest of G. We shall limit our attention, therefore, to known-segments which are PTSs. The treatment may be extended to other kinds of known-segments.

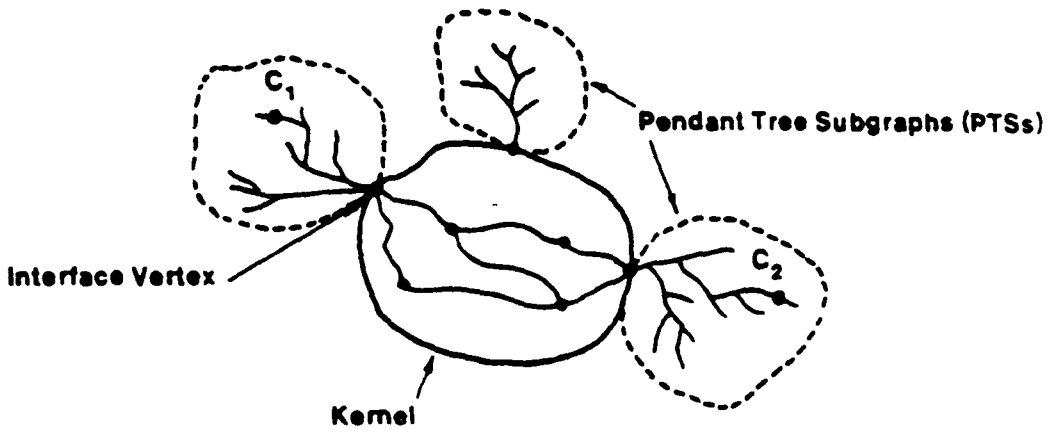
Let the set of PTS segments be $T = \{T_i(V_i,E_i)\}$ and let V_F be the set of interface vertices (see Section 6.1). We have $|V_F| = |T|$. If the $T_i \in T$ are stripped away from G, while leaving the vertices $v \in V_F$ behind, we have a graph $K(V_{kernel},E_{kernel})$, possibly unconnected and perhaps even null, whose vertex set is given by

$$V_{kernel} = V - [(\cup_{\{i|T_i \in T\}} V_i) \cup V_F]$$

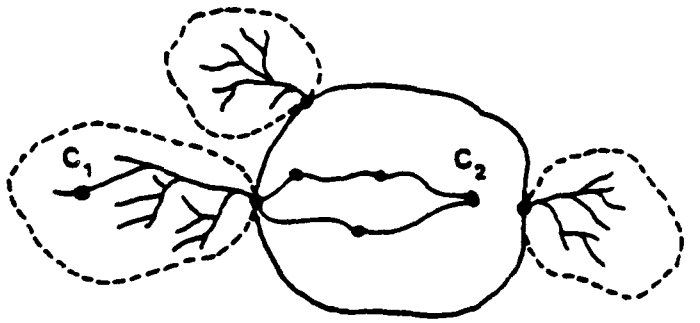
and edge set by

$$E_{kernel} = E - (\cup_{\{i|T_i \in T\}} E_i)$$

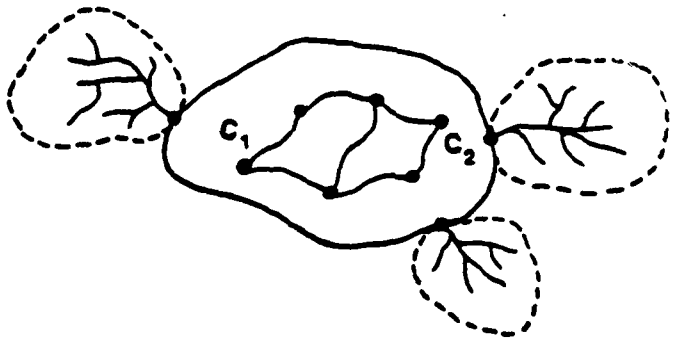
The graph $K(V_{kernel},E_{kernel})$ is defined to be the Kernel.



(a)



(b)



(c)

Figure 6-10: Three cases of paths through the Kernel

6.6.3 Paths through the Kernel

While generating an MCRS, critical components are drawn from various known-segments and perhaps also from the Kernel. In order to satisfy the Communication Axiom, all critical components in the MCRS must be able to communicate amongst themselves. The TREEREL algorithm computes the reliability of the PTSs with the assumption that all communication for components within a PTS is through the root vertex of that PTS. Assume two critical components, say c_1 and c_2, of an MCRS are in different known-segments of G (Figure 6-10(a)) and that information needs to flow from c_1 to c_2. Then it must first flow from c_1 to the root (interface) vertex of the PTS in which c_1 is contained. Thence it will enter the Kernel and flow via one or more paths through the Kernel to the interface vertex of the PTS which contains c_2. Finally it will flow from the root of that PTS to c_2 itself. If c_2 is within the Kernel (Figure 6-10(b)) the information will flow to it, after entering the Kernel, without passing into another known-segment. If both c_1 and c_2 are in the Kernel (Figure 6-10(c)) then no interface vertices are involved in the flow. Since no special techniques are known (by definition) for treating the Kernel, a path-finding algorithm is used to compute its reliability contribution for a given MCRS. The simple paths (without cycles) which are to be found are those which will enable the Communication Axiom to be satisfied.

A path between two critical components, say c_1 and c_2, is said to be a functional path if and only if each component along the path is functional. Thus the probability of a given simple path between c_1 and c_2 being functional is just the SMERGE of all the individual component success probabilities along that path. In order to satisfy the Communication Axiom at least one functional path must exist between c_1 and c_2. The probability of this event is simply the PMERGE of the probabilities of functioning of all the simple paths between c_1 and c_2.

6.6.4 The Path Algorithm

We may now describe the simple algorithm which is used to compute the probability of there being at least one functional path between a pair of components. The algorithm is recursive depth-first, uses backtracking and is quite simple-minded. It is not particularly efficient but its region of applicability, the Kernel, is a graph of fairly small size typically. Hence, its use may be tolerated. For large Kernels it may be more appropriate to use more

sophisticated algorithms such as that described by [Fratta 75].[28]

We describe the algorithm in terms of two vertices c_1 and c_2 between which a path is to be found. The vertex c_1 is the starting point, say, and c_2 is the goal. Since the algorithm is recursive, there is, at any recursive depth, one vertex on which attention is currently focused. This is called the *current vertex* and in the beginning it is c_1. At any step, the algorithm marks the current vertex as having been visited. It then checks to see if any one of the immediate neighbors of the current vertex is the goal vertex, c_2. If so, then it immediately erases the visitation mark on the current vertex and returns the (symbolic) success probability of the current vertex.[29] This probability consists of a single CRP term in whose bit vectors the bit corresponding to the current vertex is set to one. If none of the immediate neighbors of the current vertex are the goal vertex, then the immediate neighbors are checked for visitation marks. If any are marked as having been visited then a looping path has just been completed and so such neighbors are ignored. If no immediate neighbors are free of the visitation mark then the visitation mark of the current vertex is erased and a NULL is returned indicating that this was a "dead-end" and no simple paths were found. Otherwise, the algorithm is called recursively on those immediate neighbors of the current vertex which do not have visitation marks. Since each such neighbor is *potentially* the first vertex on a *different* path from the current vertex to the goal, each non-NULL value returned by recursive calls on these unmarked neighbors represents the success probability of a simple path which has been found. All such non-NULL returned values are PMERGEd since the functioning of any one of the corresponding paths would suffice to satisfy the Communication Axiom for c_1 and c_2. Finally, the result of the PMERGE is SMERGEd with the symbolic success probability of the current vertex. Then the visitation mark on the current vertex is erased and the results of the SMERGE are returned as the value of the current recursive incarnation of the algorithm.

When the recursion completely unwinds until the current vertex is again c_1, the returned value, if non-NULL, indicates the probability of the existence of at least one simple functioning path between c_1 and c_2 (If the returned value is NULL then a Communicability Edge (K-edge)

---

[28] This reference describes an elegant method for finding all the simple paths in a graph. An algebra is defined on sets of simple paths in a graph along with three path operations. This leads to the definition of a set of simultaneous linear equations the solution of which, by a method similar to Jordan's method for matrix inversion, leads to the sets of simple paths between all pairs of vertices in the graph. There appears to be a flaw in Algorithm ITER in [Fratta 75], either due to typographical errors or oversight, and this author has not been able to successfully use the algorithm in hand calculations.

[29] Note that although simple paths to the goal vertex, other than the direct edge, may exist in this case, it makes no difference whether or not they are functional if the current vertex and the goal vertex are functional since the direct edge is sufficient. Such extra paths are therefore irrelevant.

does not exist between the images of c_1 and c_2 in the K-graph G_K). The returned value is a canonical reliability polynomial (CRP) which is associated with c_1 and c_2, and the unique identities of c_1 and c_2 in G allow them to be used to derive a key to store the CRP in a hash table for later retrieval.

The procedure presented above is described in the pseudo-code for algorithm PATHREL below. The version actually used in ADVISER is slightly different in order to take into account the side constraints (see Section 6.9). Also, our graph model is undirected and so the finding of paths from c_1 to c_2 will give the same result as finding paths from c_2 to c_1. Thus the algorithm is called only once for the pair and the pair is considered unordered for computing the hash key. However, if the underlying model were to change to accommodate directed graphs, the component pair would be considered ordered and the algorithm would be used twice for each vertex pair to compute the path probability in each direction. These differences are unimportant, however, for the subject of the next section which discusses how path reliabilities are stored and used.

Algorithm PATHREL

Notation and Notes

- The functions MarkVisited(v) and UnmarkVisited(v) respectively set and remove the visitation mark on the vertex v.
- The function Neighbors(v) returns the set of neighbor vertices of vertex v.
- The function Visited(v) returns TRUE if the vertex v has its visitation mark set, FALSE otherwise.
- The function CRP(v) returns the CRP consisting of one term in which the bit corresponding to v in the NORMVEC is set. This is the symbolic reliability of the component represented by v.
- Note that the PMERGE and SMERGE functions when called with one NULL parameter, simply return the value of the other parameter.

```
Procedure PATHREL (currentvertex, goal)
Begin Local CRPaggregate;
MarkVisited(currentvertex);
if ∃i ∈ Neighbors(currentvertex) s.t. i = goal
then (UnmarkVisited(currentvertex);
    return CRP(i));
foreach i ∈ Neighbors(currentvertex)
do
  if not Visited(i)
  then CRPaggregate ← PMERGE(CRPaggregate, PATHREL(i,goal));
if CRPaggregate neq NULL
then CRPaggregate ← SMERGE(CRPaggregate, CRP(currentvertex));
UnmarkVisited(currentvertex);
return CRPaggregate
End;
```

6.6.5 The generation of partial results for the Kernel

Once G is segmented into PTSs and the Kernel, the interface vertex set V_F is known. Furthermore, when the compound requirements are provided, the number and identity of critical components in the Kernel is also known. After a potentially feasible MCRS has been chosen from the various critical components scattered over the various PTSs and/or the Kernel, it is to be determined whether the MCRS satisfies the Communication Axiom. The TREEREL algorithm implicitly computes the functioning path probability between a critical

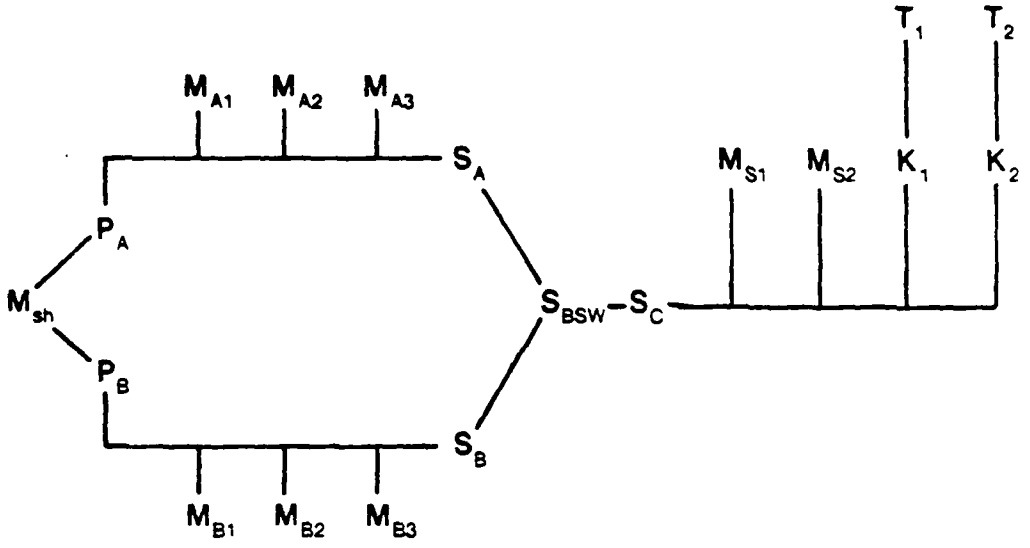
component in the PTS and its root or interface vertex. *Thus it is to be shown that the communicability graph for the interface vertices, and any critical components which happen to be chosen from the Kernel, is connected.* In other words, it must be shown for each pair of critical components in the MCRS, which may be in any one of the three cases of Figure 6-10 *in relation to one another*, that the requisite paths through the Kernel exist. This is equivalent to showing the paths exist between (a) pairs of critical components chosen from the Kernel, and (b) each critical component chosen from the Kernel and all interface vertices to any known-segment from which other critical components have been chosen.

During the reliability computation process, all feasible compositions of the integer requirements in the compound requirement will be generated over the set of known-segments and the Kernel. Thus certain path probabilities may be used repeatedly. The ADVISER program therefore computes, *a priori*, the probabilities of all necessary paths in the Kernel between <u>each</u> pair of vertices *in the set which is the union* of (1) the set of all interface vertices V_F, and (2) the set of all critical components in the Kernel. These path probabilities are hashed away in a manner very similar to the way in which PTS partial results are hashed. (see Section 6.5). Since each path probability is uniquely identified by the identities of the end vertices, it may be retrieved for use at the appropriate time. Unlike the PTS partial results the path CRPs are <u>not</u> assigned unique bits in the AUXVECs of the terms in CRPs. The reason for this is that since all the path CRPs refer to possibly intersecting sets of components (i.e. all contained in the Kernel), the complex events represented by these CRPs are, without exception, potentially dependent on one another. Furthermore, the Kernel in the case of most systems is generally composed of just a few components. Thus CRPs for path probabilities are generally short in length. It was deemed unnecessary to assign bits for these CRPs in AUXVECs only to perforce have to SMERGE them in a later phase. The path CRPs, therefore, are used directly in the generation of the system CRP.

6.6.6 The utility of side-constraints on pathfinding

It is pertinent to note at this juncture that the path-finding procedure described will, in the case of some actual PMS structures, find a communication path between a pair of components through a third component which in reality presents no such path. An example of this is the bus switch of Figure 6-11. The buses S_A and S_B may communicate with bus S_C but the bus switch, S_{BSW}, admits no direct communication between S_A and S_B. *The Kernel of Figure 6-11 is composed of the set of components*

$$\{P_A, P_B, M_{sh}, M_{A1}, M_{A2}, M_{A3}, M_{B1}, M_{B2}, M_{B3}, S_A, S_B, S_{BSW}\}$$



Key

M_{sh}	Dual-Ported Shared Memory
M_{A1}, M_{A2}, M_{A3}	Local memory, Bus A
M_{B1}, M_{B2}, M_{B3}	Local memory, Bus B
S_A, S_B	Processor Buses
S_C	External Bus
S_{BSW}	Bus Switch
M_{S1}, M_{S2}	Secondary Store
K_1, K_2	Device Controllers
T_1, T_2	Peripheral Devices

Figure 6-11: A dual-port bus-switch architecture

wherein S_{BSW} is the sole interface vertex. We do not wish the path finding algorithm to find the path (P_A S_A S_{BSW} S_B P_B) between the critical components P_A and P_B in the general case. In this particular case, since S_{BSW} is an interface vertex of the Kernel, if M_{S1} were a critical component, then the finding of the path (P_A S_A S_{BSW} S_B P_B) would not matter since S_{BSW} would appear in all system success states anyway. In Section 6.9 we shall propose a set of side-constraints which may be imposed on the PMS structure to be analyzed so that ambiguities of this and other sorts can be resolved.

6.7 The Main Loop of the Overlord Routine

The *Overlord routine* in ADVISER controls all of the actual assembling of the system reliability functions for a given compound Boolean requirement expression. In general, the compound requirement can be a disjunctive requirement. If this is so, the Overlord routine expresses it as a disjunction of purely conjunctive requirements (i.e. the "sum-of-products" canonical form for a Boolean expression). Then a CRP is derived for the PMS structure for each of the conjunctive requirements in this sum-of-products form. Finally, the CRPs for all the conjunctive requirements are PMERGEd to obtain the CRP for the disjunction.

The main loop of the Overlord routine accepts a purely conjunctive requirement and returns a CRP which is the reliability function of the PMS structure under that conjunctive requirement. Each conjunctive requirement is decomposed by the Overlord routine into the atomic requirements which comprise it. In general, in the instance of any one of these atomic requirements, say $\psi(r_i,x_i)$, the PMS structure may have $u_i \geq r_i$ components of type x_i. There are then $\binom{u_i}{r_i}$ ways of satisfying $\psi(r_i,x_i)$. There are, therefore,

$$\prod_i \binom{u_i}{r_i} \quad u_i \geq r_i$$

ways of satisfying the conjunctive requirement. However, the u_i components of type x_i will, in general, be scattered throughout the various known-segments and/or the Kernel of the PMS structure. Hence there is an upper bound on the number of components of type x_i that a given segment can contribute toward satisfaction of the requirement. The Overlord routine calls the *TREEREL and PATHREL algorithms* to compute the reliability contributions, expressed as CRPs, for each of the segments, for each atomic requirement in the conjunctive requirement. for each possible number of components chosen from that segment to satisfy the requirement. The number of components chosen from a segment varies from unity up to either the upper bound alluded to above or to n_i, whichever is smaller. The CRPs. thus derived, are hash-coded away for later retrieval and use. Such hash-coding obviates the need for repeated recomputation of the identical CRPs several times over the course of a program run.

The main loop of the Overlord routine utilizes these hash-coded partial results while constructing the CRP of the PMS structure under the conjunctive requirement. For each iteration through the loop, the sequence of steps described below is executed. The results of the iterations are accumulated and the accumulation, after the final iteration, represents the CRP under the conjunctive requirement. The steps are first described broadly. Following

sections will provide details on the steps. Through the rest of this section (Section 6.7) we shall interchangeably use "component" and "type" for "critical component" and "critical component type" respectively.

1. *Generate next composition of the requirement integers over the known-segments and the Kernel (if no more compositions can be generated, go to Step 6)*: Compositions were described on Page 125 and in Chapter 2. Each composition here represents one possible case of satisfaction of the requirement. A composition specifies what number of each required (critical) type of components are to be chosen from each known-segment and the Kernel. Only feasible compositions (see Page 127) emerge from the composition generating function. Feasible compositions are those which do not demand that more components of any type be chosen from any segment than are present of that type in the segment.

2. *Determine if the Kernel will satisfy the Communication Axiom for this feasible composition. If not, go to Step 1*: Computes the CRP which represents the contribution of the Kernel for this feasible composition.

3. *For each known-segment, and for each component type, retrieve the CRP from the hash-tables which represents the reliability contribution of the known-segment in the case that the number of components of the type, specified by the current composition, are chosen from it.*: The CRP may, of course, be null in the case that there are no components of a particular type in the segment.

4. *SMERGE the CRPs retrieved in Step 3 with the Kernel CRP of Step 2*: The SMERGE operation of this step accounts for the fact that all the known-segments and the Kernel must simultaneously satisfy the various requirements imposed on them by the current composition.

5. *Accumulate the result of Step 4 by PMERGEing it into the accumulation thus far. Then go to Step 1*: This step accounts for the fact that the satisfaction of any one composition provides a reliable system. Hence we must take the disjunction of the CRPs for satisfied compositions.

6. *The accumulated CRP at the end of the iteration over Steps 1 through 5 represents the reliability of the PMS structure under the conjunctive requirement.*

6.7.1 Generation of feasible compositions

The implementation of the process of generation of feasible compositions will be summarized here. The process is also introduced in Chapter 2 and treated at length in Section 6.4.

The ADVISER program maintains a two-dimensional array called the Compositions Table which it uses in the generation of compositions (see Figure 6-12). In addition, a one-dimensional Requirements Array is maintained parallel to the columns of the Compositions

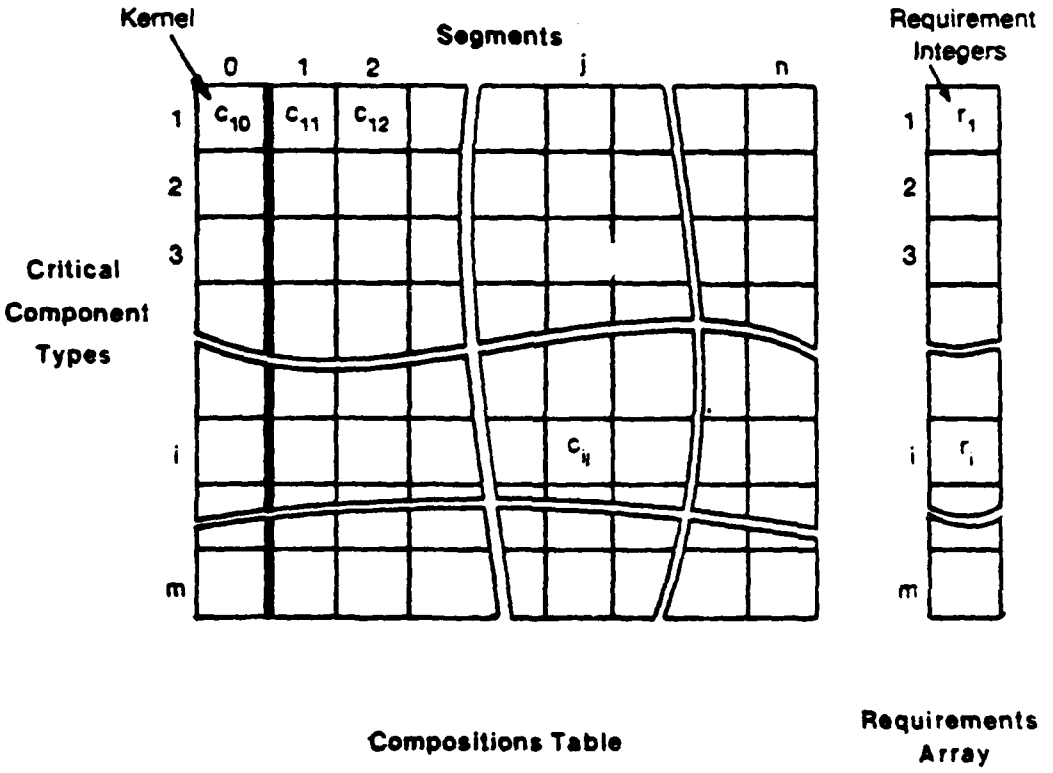


Figure 6-12: The logical organization of the Compositions Table

Table. The rows of the latter correspond to all the distinct critical component types which are specified in the conjunctive requirement. Each column of the Compositions Table corresponds to a known-segment of the PMS graph, except the zeroth column which corresponds to the Kernel. For a given conjunctive requirement

$$\bigwedge_{1 \le i \le m} \psi(r_i, t_i)$$

the contents of the i^{th} cell of the Requirements Array will always hold the integer requirement r_i. The contents of the cells of the Compositions Table are subject to change each time a new feasible composition is generated. It is always the case that the i^{th} row of the Compositions Table holds some $(n+1)$-composition of the requirements integer r_i in cell i of the Requirements Array. If each particular distribution of integers in the cells of the Compositions Table is considered a state of the Table, then each state of the Table is a set of feasible compositions in the rows, after control emerges from the composition generator function. Such a state will be termed a feasible state of the Table. However, as was pointed out earlier in this chapter even though a Compositions Table state is feasible, it will not contribute to system reliability unless the Communication Axiom is satisfied. The contents, say c_{ij}, of cell [i,j] of the Compositions Table in any particular state, specifies that c_{ij} components of type i must be chosen from known-segment j (or the Kernel if $j=0$). Thus in Step 3 above, for known-segment j, $j \neq 0$, the program advances down column j and for each c_{ij} it retrieves the hash-coded CRP for the atomic requirement $\psi(c_{ij}, t_i)$ on that segment (i.e. PTS). Of course, if $c_{ij}=0$, then the CRP is null. Likewise, if no critical components are chosen from a given known-segment then the corresponding column of the table will contain all zeroes and will be ignored. The zeroth column of the Compositions Table is passed to the DoCore function which computes the CRP for the reliability contribution of the Kernel and ensures that the Communication Axiom is satisfied. This function is described in Section 6.7.2.

As noted above, there may be an upper bound,say u_{ij}, on components of type t_i which can be chosen from segment j such that $u_{ij} < r_{ij}$. One may, therefore, think of an "upper-bound state" of the Compositions Table which constrains the c_{ij} values. Some of the $(n+1)$-compositions of a given r_i may not be useful if some cell of the i^{th} row contains an integer which is greater than u_{ij}, its upper bound. Thus, a state of the Compositions Table is feasible if and only if

$$\forall i,j \; c_{ij} \le u_{ij} \qquad (6.2)$$

The generating routine for the next feasible composition enumerates all the possible states of the table but returns only those which satisfy condition (6.2) above. Since the i^{th} row of the table will produce

$$\binom{(n+1)+r_i-1}{r_i-1}$$

separate $(n+1)$-compositions of the integer r_i, the total number of states (feasible or infeasible) of the Compositions Table is

$$\prod_{i=1}^{m} \binom{(n+1)+r_i-1}{r_i-1}$$

This can be a large number but the process of generating each state is incremental and thus fast. The upper bound check does not add much more complexity. Moreover, as will be described in Chapter 7, the largest fraction of compute time during a run of the ADVISER program has been experimentally observed to be consumed in another portion of the program.

As was noted in Section (6.1) there is a strong analogy between the action of an odometer and the generation of feasible states of the Compositions Table. Each row of the Table corresponds to a wheel of the odometer. The compositions which may occupy the cells of a given row are analogous to the numbers on the corresponding odometer wheel. Therefore, one complete revolution of the wheel corresponds to the generation of one complete cycle of $(n+1)$-compositions in the row. Row 1 of the Table corresponds to the slowest moving wheel while row m corresponds to the fastest moving wheel of the odometer in our analogy. However, since only feasible states of the Compositions Table are ever used, the odometer may be viewed as having slippage on some of its wheels. This would cause the odometer to skip those positions which correspond to the infeasible states of the Table.

With this odometer analogy in mind we may view the actual generation of all possible states of the Compositions Table which treats the table as a stack. Each row of the Table in the implementation scheme corresponds to a level in the stack and the m^{th} row (see Figure 6-12) is at the top of the stack. In other words, the next $(n+1)$-compositions of the requirement integer r_i at the i^{th} level of the stack (i.e. the i^{th} row) is computed only after all possible $(n+1)$-compositions of the integer r_{i+1} have been computed at the $(i+1)^{th}$ level. Also, whenever the i^{th} row is advanced to the next $(n+1)$-composition of r_i,[30] all the rows $(i+1)$ through m are reset to their initial $(n+1)$-compositions. An initial composition for the i^{th} row consists simply

[30] The algorithm used to generate the next composition is a variant of the one described in [Nijenhuis 78] with modifications to do the upper bound checks and return only feasible compositions.

in putting the integer r_i in the zeroth cell of the row.[31] The process ends when all $(n+1)$-compositions have been exhausted at row 1.

6.7.2 Computing the reliability contribution of the Kernel

The process of computing the reliability contribution of the Kernel, carried out by the DoCore function in ADVISER, plays a critical part in deciding whether a feasible state of the Compositions Table will actually produce system success. The decision is based primarily on whether the Communication Axiom can be satisfied by the current state of the Compositions Table (recall that each state of the table corresponds to one particular way of choosing critical components from the various parts of the graph to satisfy the overall requirement). The decision also depends on any side-constraints which may have been specified. The side-constraints are not important to the elucidation at this time and a discussion of them is deferred to Section 6.9. Any feasible state of the Compositions Table which passes the check by the DoCore function is termed a success state of the Compositions Table.

The reason that the check for the satisfaction of the Communication Axiom is localized to the Kernel, and not the known-segments, lies in the difference between the algorithms used on the two kinds of subgraphs. The TREEREL algorithm assumes that all communication between components in the tree, and to other components in the rest of the graph, is through the root vertex. The recursive descent nature of the algorithm starting from the root vertex ensures that the probability of functioning of paths to the root is accounted for in the case of all the critical components chosen from the tree. Hence the CRP returned by the TREEREL algorithm also accounts implicitly for satisfaction of the Communication Axiom as far as communication between the root and other components in the tree is concerned. As a result it falls to the DoCore function, which treats the Kernel, to check whether the Axiom is satisfied by the Composition Table state being considered.

We now digress to introduce terminology which will make a description of the DoCore function clearer. For each state of the Compositions Table some fragment requirements will be applied to some or all of the known-segments and the Kernel. Those known-segments which do not have fragment requirements applied to them will be termed currently dormant known-segments in the given state. The other known-segments, against which fragment

---

[31] In practice, however, it is sometimes possible to start off a row with some intermediate composition, which it would have ultimately reached in the normal course, because the constraints placed on the compositions in that row preclude previous compositions from being feasible.

requirements *have* been applied, will have some critical components chosen from them. Such known-segments will be termed currently active segments. The Kernel will be currently active, in the same sense, when either critical components are chosen from it, or paths must exist in it which link the root vertices of active known-segments, or both these conditions hold. Note that it is possible for the Kernel to be dormant in the above sense when some known-segment by itself alone is able to satisfy the entire conjunctive requirement. Then the TREEREL algorithm applied to that PTS will implicitly ensure satisfaction of the Communication Axiom and provide the reliability contribution of the corresponding Compositions Table state[32]. The DoCore function will not be invoked in such an instance since paths through the Kernel are *not involved*.

In a given Compositions Table state, when the Kernel is currently active, attention is directed toward two kinds of vertices within it, namely

1. Interface vertices of active known-segments, and

2. Critical components chosen from the Kernel in the current state.

The set of vertices in Items 1 and 2 together will be termed the Currently Chosen Kernel Set (CCKS). Note that for a given Compositions Table state there may be many CCKSs. The reason for this is understood by considering the following example. Assume that b_1 components of critical component type t_a are present in the Kernel and that the current state of the Compositions Table requires that b_2 components ($0<b_2 \leq b_1$) of type t_a be chosen from the Kernel. There are then $\binom{b_1}{b_2}$ ways of doing this, each of which will produce a different CCKS.[33] It is important to note, however, that the set of *interface* vertices, contained in these CCKSs of the same state, does not vary since the set of currently active known-segments does not change. Also to be noted is that since the reliabilities of the interface vertices are taken into account during the computation of the PTS reliabilities, they are not included in the path reliability calculations.

Returning to our discussion of the DoCore function we now note the following important point. Since the Communication Axiom is implicitly satisfied within known-segments due to the action of the TREEREL algorithm, *the test for satisfaction of the Axiom, by the entire PMS structure, may be confined to the CCKS.*

---

[32] However, see Section 5.3 for a current deficiency in the TREEREL algorithm which could cause an error here.

[33] This situation also arises in known-segments. But the TREEREL algorithm returns a CRP which takes into account all the possible cases. In the Kernel, however, the enumeration must be done explicitly.

Using the terminology of Section 6.6.1 we may restate the above condition as follows: *The Communication Axiom is satisfied by the PMS graph G if there exists a connected K-graph on the critical components chosen from the Kernel and the interface vertices of those known-segments from which at least one critical component was chosen.*

As was described in Section 6.6.4 the PATHREL algorithm returns a CRP if a K-edge exists between a pair of designated components and null otherwise. Each such CRP was then hash-coded, keyed on the identities of its end vertices. Thus, for any given pair of components in the Kernel, if the key is computed and no associated CRP is found in the hash table then no K-edge exists between those two components. The task of the DoCore function then reduces to examining all pairs of components in the CCKS and attempting to retrieve a CRP from the hash table for each pair. The number of K-edges thus retrieved are counted and must number at least one less than the cardinality of the CCKS: this is a necessary condition for connectedness of the K-graph. Furthermore, each vertex in the CCKS must be connected in the K-graph to at least one other component in the CCKS. This ensures connectedness of the K-graph. Once connectedness is established, then the CRPs of all the K-edges are SMERGEd together to give the CRP for the CCKS. If the K-graph is not connected then the next CCKS is generated and the process continues until all possible ways of choosing the required number of critical components from the Kernel have been considered for the current Compositions Table state.

Since any one of the CCKSs which satisfies the Communication Axiom will make for system success, the CRP of the Kernel for a given state of the Compositions Table is the PMERGE of all the CRPs of the CCKSs of that state and the DoCore function returns the result of this PMERGE as the reliability contribution of the Kernel.

6.7.3 Computing the reliability contribution of the PTS segments

The contribution of the PTS segments (or known-segments) towards the system reliability were precomputed and hash-coded away. They are now retrieved for use. It will be recalled that the hash-code keys in this case were based on the requirement integer, the required component type and the identity of the root vertex of the PTS. For any state of the Compositions Table, a PTS segment j corresponds to column j of the table, the required component type t_i corresponds to row i and the integer requirement r_i is the contents of the cell at the intersection of that row and column i.e. c_{ij}. Thus the key for the CRP of the segment j under the atomic requirement $\psi(c_{ij},t_i)$ can be computed and the CRP may be retrieved for use. Note that (i) if $c_{ij} = 0$ then the CRP is null, and (ii) if $c_{ij}>0$ then the process of generating

partial results for PTSs (see Section 6.4.3.1) guarantees that a CRP will have been hash-coded away.

Since we are dealing with purely conjunctive requirements, all the atomic requirements specified by the cells of the Compositions Table (other than in column 0, which refers to the Kernel and is treated separately by the DoCore function) must be satisfied simultaneously. The CRP for this event is obtained by SMERGEing all the PTS segment CRPs retrieved for the current state of the Compositions Table.

6.7.4 Accumulating the result for a pure Conjunctive Requirement

We have seen above how CRPs are constructed for the reliability contribution of the Kernel and the combined reliability contribution of the PTS segments, for each state of the Compositions Table which is capable of satisfying the Communication Axiom. Since both the Kernel and the active known-segments must simultaneously be functional, we must SMERGE the final CRPs obtained as described in Sections 6.7.2 and 6.7.3 above. This gives the CRP for the event that the current state of the Compositions Table is a success state of the table i.e. gives rise to one or more system success states.

Now, any state of the Compositions Table which allows system success states will contribute toward system reliability. Thus, finally, the CRPs for all success states of the Compositions Table must be PMERGEd to obtained the CRP for the PMS structure under the overall conjunctive requirement.

6.7.5 General case: a Disjunctive Requirement

Recall further that if the user of ADVISER supplies a general disjunctive requirement, then this is rephrased as a disjunction of purely conjunctive requirements and the Overlord routine main loop is called once for each of these. For each of the conjunctive requirements a CRP will be returned by the main loop as outlined in Section 6.7.4. Since the satisfaction of any one or more of these conjunctive requirements implies system success, all the CRPs, returned by the various cases to the main loop with conjunctive requirements, are PMERGEd to provide the CRP which gives the system success probability under the general disjunctive requirement expression.

6.8 Efficiency in the assembling of CRP's in Overlord

It will have been noted in the foregoing sections that the PMERGE and SMERGE operations are frequently used. Furthermore, it was shown in Chapter 3 that the time complexity of the SMERGE operation is $O(N^2)$ whereas that of the PMERGE operation is $O(N^2 + 2N)$ where N is the length of the CRP lists being operated upon. Depending on the complexity of the PMS structure, and the input requirements expression, the length of CRPs relating to states of the Compositions Table begins to get rather large. Lengths on the order of 1000 and more terms have been observed by the author in experiments. Hence one may predict that ADVISER would spend the largest part of its compute time in the PMERGE and SMERGE operations. This is indeed strongly borne out by experience, so much so that in most reasonably complex cases of PMS structures and requirements the percentage of compute time taken by the PMERGE and SMERGE operations over the run of the program, largely outweighs all other costs of other computations during the run. Chapter 7 shows that in a fairly typical case runtime consumed in the merge package of the current ADVISER implementation could range as high as around 88% of the total compute time. Hence it is imperative that the number of PMERGE and SMERGE operation be reduced as much as is possible if the current *intermediate representation and its algorithms continue to be used.*

Referring back to the Compositions Table we see that because of the stack discipline while generating the states of the table, all rows in the table, except for row 1, cycle through their $(n+1)$-compositions more than once. Thus, the CRP corresponding to some $(n+1)$-composition of some row i, $2 \leq i \leq m$, will be used in an SMERGE operation more than once. Each time, the accumulated CRPs of Steps 4, 5, and 6, on Page 150, may, and usually will, grow in length. Thus each succeeding merge operation on the accumulated results takes more and more compute time. There is clearly incentive here to keep the CRP term list lengths as small as possible. The problem may be alleviated considerably by postponing the mergings of Steps 4 and 5, on Page 150, until a later phase when some economies may become apparent.

The stack discipline of the Compositions Table suggests a remedy. The PMERGE and SMERGE operations are associate and commutative over one another. Thus the CRP for a given state of the Compositions Table, may be arrived at by simply SMERGEing all the CRPs for all cells, not in column 0 of the table, in any desired order. The result of this is then SMERGEd with the Kernel CRP for the table state as computed by the DoCore function using column 0 of the table and other criteria such as the Communication Axiom. However, due to the stack discipline of the table, the CRPs for the cells of row i will not change until row i + 1

has cycled through all of its (n + 1)-compositions. Thus, a savings might be effected by postponing the SMERGEing of the CRPs of row i to a later time so that it may be done only once. We ignore the zeroth column of the table when speaking of the rows at this time, because of the special treatment accorded to it by the DoCore function, and concentrate on columns 1 through n. We shall term as a Row-CRP that which is produced by SMERGEing the CRPs retrieved by keying on the values of the cells of any row (column 0 cell excluded). Similarly, we shall term as the Kernel-CRP that which is produced by the DoCore function operating on column 0. Clearly, the CRP of any state of the Compositions Table is the SMERGE of all the Row-CRPs of the state, SMERGEd with the Kernel-CRP. However, the CRP for the conjunctive requirement may be computed more efficiently than this as is described in the following.

6.8.1 The CRPTree

We now expand on the notion of the CRPTree which was introduced in Section 6.4.3.2. The CRPTree is a rooted tree of (m + 2) levels where m is the number of critical component types referenced in the conjunctive requirement. This is also the number of rows in the corresponding Compositions Table. The root vertex of the CRPTree is a dummy vertex. The root will be said to be at level zero of the CRPTRee. All vertices at level i, $1 \leq i \leq m$, of the tree will correspond to row i of the Compositions Table (excluding column 0). The leaf vertices at level m + 1 will correspond to the Kernel (i.e. to column 0 of the table). Figure 6-13 shows a partial instance of a CRPTree. Each vertex at level i corresponds to some distinct (n + 1)-composition in row i of the Compositions Table and the vertex is labelled with the Row-CRP of that particular (n + 1)-composition. Each vertex at level i, $1 \leq i \leq (m-1)$, can have at most

$$\binom{\quad}{\quad}$$

successor vertices. This number is the total number of *possible* (n + 1)-compositions (of the requirement integer r_{i+1}) which can occupy row i + 1. Thus the tree has, in the most general case, a different maximum branching factor at each level, which is equal to the total possible number of (n + 1)-compositions assumed by the row corresponding to the next lower level. Each path from the root vertex of the CRPTree to a vertex at level m, therefore, represents a set of CRPs which are the labels of the vertices along the path. Each such set of CRPs is precisely the set of all Row-CRPs for a particular state of the Compositions Table. Finally, each vertex at level m will have precisely one pendant successor vertex at level m + 1 whose label is the Kernel-CRP for the state corresponding to the path from the root to that level-m vertex. Therefore, each path in the CRPTree from the root vertex to a leaf vertex corresponds to a unique success state of the Compositions Table.

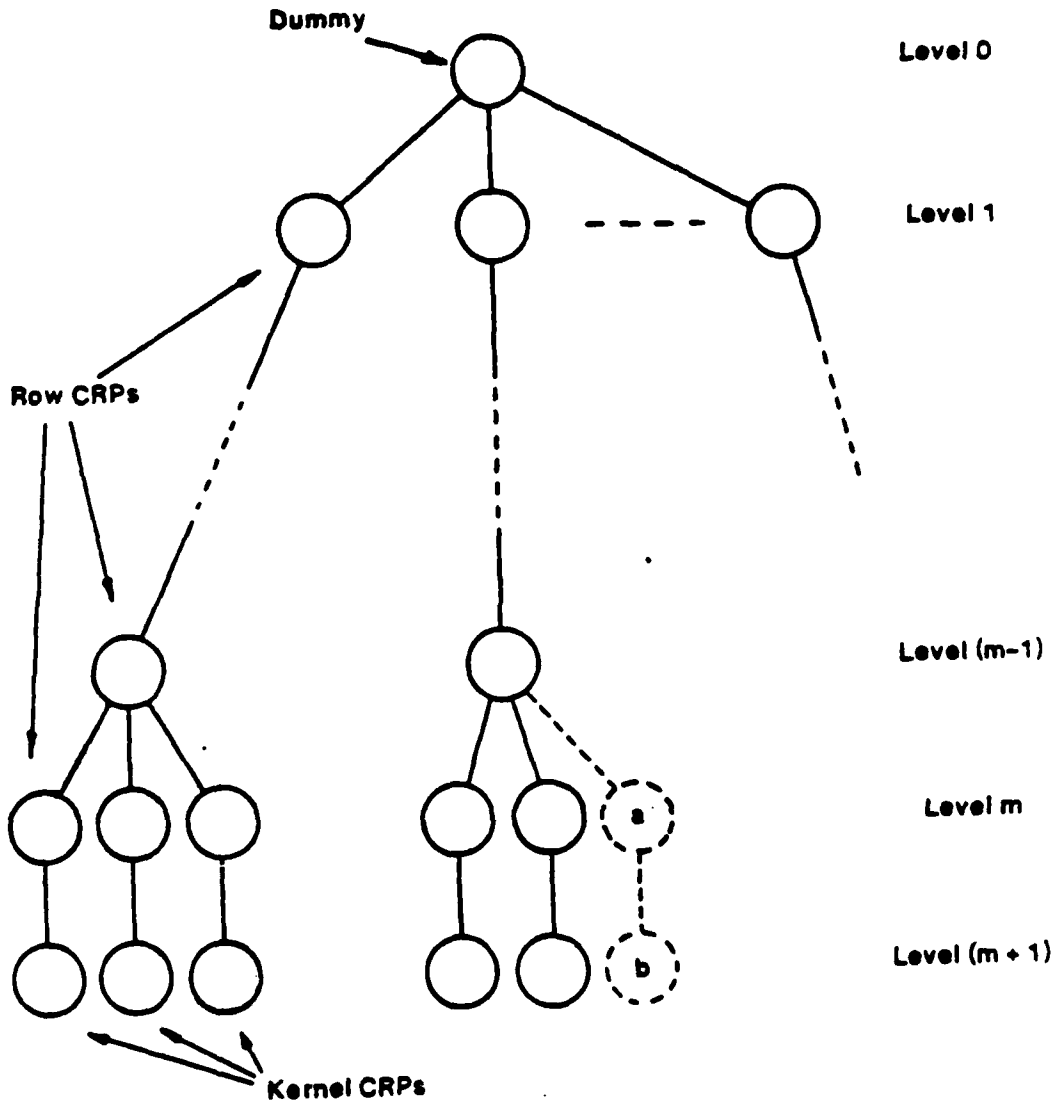


Figure 6-13: An example of a CRPTree

In the extreme case, every state of the Compositions Table will contribute to system reliability, and the DoCore function will return a non-null CRP after some computations for each such state which use column 0 of the table. Then the tree will be complete at each level to the maximum branching factor at that level. If the $(m+1)^{th}$ level (i.e. Kernel-CRPs) are ignored, this complete tree will also be symmetric in that the sets of labels (CRPs) of successor vertices, for all vertices at a given level will be identical. Unfortunately, the Kernel-CRPs may in general be distinct from each other thus destroying the symmetry of the tree. However there may still exist some symmetric subtrees of the overall CRPTree when the Kernel-CRPs at their corresponding leaf vertices are identical.

In a more typical case, not all states of the Compositions Table contribute to system reliability. Excluded would be those states for which the Kernel-CRP is null due to the inability of the Kernel to satisfy the Communication Axiom. The exclusion of a Compositions Table state corresponds to removing the leaf vertex, which would have been labelled with the Kernel-CRP for that state, and all vertices on the path to that leaf which are not shared with other paths, e.g. vertices a and b in Figure 6-13. Thus, in the typical case, the tree may become incomplete at all levels. The lack of symmetry in the tree is a hindrance to efficient computation but it may be possible to use what little symmetry still exists.

6.8.2 Construction of the CRPTree

The CRPTree is quite simply constructed in a recursive fashion during the main loop execution in the Overlord routine. For any row i, $1 \leq i \leq m$, in the Compositions Table, a vertex at level i of the CRPTree is produced when the contents of the row's cells advance to the next $(n+1)$-compositions of r_i. Each $(n+1)$-composition is held constant in row i until row $i+1$ has cycled through all its $(n+1)$-compositions of r_{i+1}, thereby producing a set of vertices at level $i+1$. *These vertices are made successors of the vertex generated for the current $(n+1)$-composition at level i.* Once row i has cycled through all of its $(n+1)$-compositions of r_i, the set of level-i vertices generated during the cycle is passed upward to row i-1 to become successors to the current vertex at level i-1. The vertices generated for row 1 are made successors of a dummy vertex labelled with a null CRP and designated the root vertex. Each row may be viewed, therefore, as passing a set of subtrees of the CRPTree to the immediately previous row each time it cycles through all its $(n+1)$-compositions.

The $(m+1)^{th}$ level of the tree consists of vertices labelled with Kernel-CRPs. A level-$(m+1)$ vertex of the tree is generated whenever the DoCore function returns a non-null CRP after operating on the contents of column 0 in the current state of the Compositions Table. The

generation of a level-(m + 1) vertex signifies that the current state of the Compositions Table satisfies the Communication Axiom and contributes to system reliability.

6.8.3 Use of the CRPTree

The purpose of building the CRPTree is to postpone the bulk of the PMERGE and SMERGE operations to a phase after the completion of the main loop execution in the Overlord routine. At that time, as explained above, using the CRPTree as the data structure fewer of the merge operations need be done to complete computation of the system reliability function. Furthermore, it may be possible to use any symmetry in the CRPTree to advantage by doing the indicated merge operations for one of the symmetric subparts and using the resultant CRP as a template for the rest.

The procedure for computing the CRP of any subtree t_s of the CRPTree is simply stated in a recursive fashion as follows:

1. The CRP of any subtree t_s of a CRPTree is obtained by
   a. PMERGEing the CRPs of the subtrees rooted on the successor vertices of the root vertex of t_s, and then
   b. SMERGEing the resultant CRP of Step 1 with the CRP which labels the root vertex of t_s.
2. The CRP of a one-vertex subtree is simply the CRP which labels that vertex.

A simple implementation of this is the recursive procedure CRPTREEMERGE which is shown on Page 135.

6.9 Side Constraints on Reliability Function generation

We have seen that at least three basic items of information are necessary to compute system reliability, namely

1. Reliabilities of individual components in the system,
2. The interconnection topology of the system, and
3. Minimum task requirements on component reliability which determine the system reliability in relation to the task.

Thus far we have considered PMS structures to be mapped into undirected graphs with labelled vertices. The implicit assumptions regarding possible paths of information flow in the

structure have been precisely those which are made in classical network reliability analysis. To wit, information may flow into a vertex (component) from any arc incident on the vertex and exit it from any other of its incident arcs. Thus typical network reliability analysis examines concepts such as the probability that a particular vertex will always be able to communicate with some other specific vertex for the duration of the mision; the probability that a network of a certain diameter will be reliable over the mission time, etc.

The kinds of questions which arise out of the analysis of Processor-Memory-Switch structures, however, concern themselves with a *minimum working set* of components which must be functional and able to communicate amongst themselves in order for the system to be reliable. Qualitatively, a PMS network differs from general communication networks in the degree of coupling between system components; the coupling is much tighter in PMS structures. Another fundamental difference is the intuitive model of behavior of a node in a communication network and a node in a PMS structure. In the latter case the nodes in the structure cannot usually be considered to be homogeneous and their internal information flow characteristics are not uniform . As was shown in Section 6.6.6, treating PMS nodes as being able to transfer information from any incident arc to any other incident arc can lead to *incorrect reliability estimates*. Furthermore, the behavior of actual PMS level components such as buses, memories, processors, etc., is not adequately modeled.

It is clear, then, that further constraints beyond the three stated above must be imposed on the problem in order to obtain an adequate reliability model. At the same time, in a program that is viewed as an estimation tool for design, it is impossible to incorporate information about every type of PMS component that exists or may exist in the future. Thus an effort was made to distill those aspects of the information flow characteristics of PMS level components and commonly occurring PMS substructures, which when combined with the Communication Axiom would provide adequate reliability models in a majority of instances. For example, the adoption of Pendant Tree Subgraphs as the sort of known-segments to achieve during the problem partitioning, was driven by the observation that PTSs occur in a large number of PMS structures; typically in the input/output subsystems and bussed architectures.

The PMS level of detail at which systems are studied [Bell 71] is characterized by a lack of information specific to the behavior of each system component. Rather, the emphasis in modelling at the PMS level is directed toward the system interconnection structure and broad details on the rates and types of information flow among the system components. To paraphrase Bell and Newell, this is the chemical engineering view of computer systems. In keeping with this view, and in an attempt to preserve generality, the side constraints are based

on system interconnection structure and pathways of information flow among components rather than the specific behavior of certain types of system components.

In succeeding sections we shall examine three kinds of side constraints on the reliability function generation process which, when judiciously applied, will ease the task of the program and provide more accurate models. For each case we discuss, in order

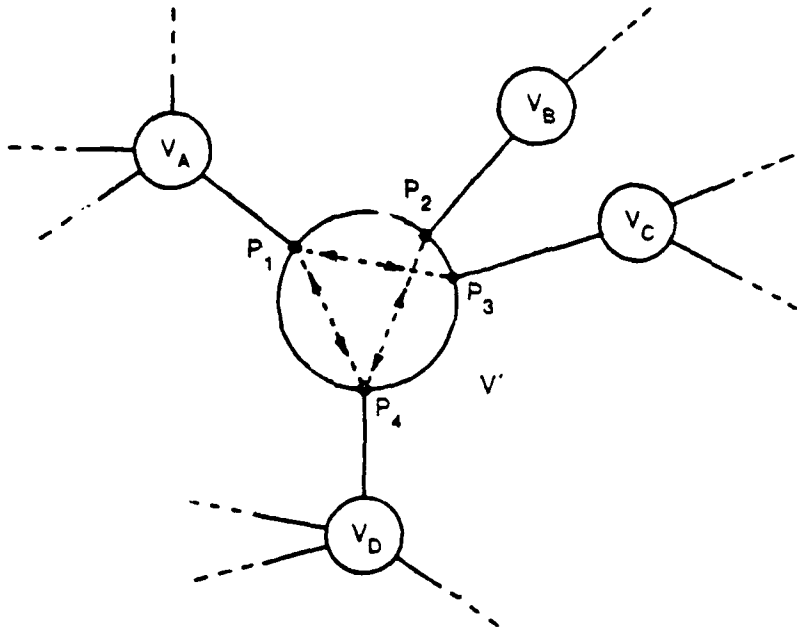
- the need for being able to specify the constraint,
- the implementation of the constraint, and
- the changes which are necessary to other algorithms to be able to deal with the constraint.

6.9.1 Intracomponent port connections

6.9.1.1 Need for constraint

As was stated above, the classical model applied to a node in a communication network constrains its behavior very weakly. In principle, such a node is deemed capable of accepting incoming information flow from any incident arc and transmitting it out over any other, or the same, incident arc. Section 6.6.6 gave the Bus-Switch architecture as an example wherein there is a component in the PMS structure which is incapable of such general behavior. Consider the model in Figure 6-14(a) where a vertex v′ is shown enlarged and its neighboring vertices are v_A, v_B, v_C, v_D. The P_i are the connection ports of the component represented by v′. The broken lines inside the circle representing v′ correspond to the possible paths of information flow within the component. They, and the arrows on them, show the possible path and direction of information flow between their end ports. Thus, for instance, information flowing in from port P_2 could, after processing within v′, exit only via port P_4, and *vice versa*. The ports themselves do not act as intermediate stopping points on such paths i.e. the existence of the paths (P_1,P_4) and (P_2,P_4) does not imply that information may flow from P_1 to P_2 via P_4.

If the internal port connectivity of v′ were not known, it would be surmised that information could flow from say v_A to v_B although the component behavior does not support this. Thus, v′ may become part of a "functioning" path even though v_A and v_B are not allowed to communicate directly through v′. Many cases of such behavior may be noticed in PMS structures e.g. the Bus-Switch architecture of Section 6.6.6. The internal port connection constraints attempt to account for such behavior in general fashion.



(a)

V'

	P_1	P_2	P_3	P_4
P_1	1	0	1	1
P_2	0	1	0	1
P_3	1	0	1	0
P_4	1	1	0	1

Internal Port Connection Matrix

(b)

Figure 6-14: An example of a vertex with an Internal Port Connection Matrix

6.9.1.2 Implementation

The most natural way of specifying the intra-component port connection constraint is in the form of an Internal Port Connection Matrix (IPCM). This is simply an adjacency matrix for the ports of the component. The IPCM for the vertex v' of Figure 6-14(a) is shown in part (b) of the figure. It may be noted that the matrix is symmetric. This is a direct consequence of our basic model of the PMS structure as an undirected graph. Were the underlying model changed to support directed graphs then the IPCM would not necessarily be symmetric. *In the ADVISER program, the absence of an IPCM associated with a vertex is assumed to imply that all elements of the IPCM, were one to be appended to the vertex, are unity, i.e. that each port can transfer information to and from any other port.*

For the current version of ADVISER the IPCM for any given vertex is specified only after the interconnection graph has been specified (a preferable method is described below). The vertex is identified by naming the component it represents and the value of the [i,j]th element of the IPCM for the vertex is set to 1 by naming the pair of neighboring components of the vertex which are connected to the i^{th} and j^{th} ports respectively. For the current graph model, all diagonal elements of the IPCM default to 1 and all others default to 0 if not set to one. Clearly, this approach of setting elements of a given IPCM to unity can become tedious if the number of elements to be so set is large or the number of IPCMs is large. Neither case may be expected to be common in typical PMS structures. Furthermore, IPCMs need only be specified for those components wherein the default complete internal interconnectivity of ports would lead to erroneous paths being discovered by the path-finding algorithms. Thus, the current method of specification was deemed adequate though tentative.

A better method of specification would be to allow the particular IPCM to be associated with a generic component type. Then when a component of that type was instantiated in the PMS structure the IPCM would be automatically declared. However, the connection of other components to this component would have to be done more carefully, keeping port identities in mind. There would also arise issues of what is to be done in case the user of the program does not connect any components to a certain port. These questions are left to a future implementation update of ADVISER.

6.9.1.3 Effect of constraint on algorithms

The Intra Component Port Connection constraint affects the operation of two of the algorithms in ADVISER. The first, and most affected, is the PATHREL algorithm (see Section 6.6.4). It is no longer sufficient, at each recursive call to PATHREL, to simply check the visitation mark on a neighbor vertex when deciding whether or not to recursively call the procedure on that neighbor vertex. The reason for this is that the current vertex may not internally allow a path from the previous vertex to the neighbor vertex being considered. Returning to Figure 6-14, presume that a call of PATHREL on the vertex v_A may call a recursive incarnation of the procedure on its neighbor v'. The incarnation on v' should not consider v_B as a candidate for the next recursive call since there is no path from v_A to v_B through v'. Thus an extra parameter is added to the PATHREL procedure which "remembers" the identity of the vertex out of which the current recursive call arose, i.e. the originating vertex. Going back to our example, the current incarnation of PATHREL on v' would have an "originating vertex" parameter whose value is v_A. The algorithm when considering a neighbor vertex for a recursive call will first look up the IPCM of the node being currently visited. It will see if the element corresponding to the originating vertex and the currently considered neighbor vertex can communicate through the currently visited vertex, i.e. if the corresponding IPCM element is unity. If the IPCM element is zero then that neighbor vertex is not visited even if it has no other visitation marks.

The other algorithm affected by this constraint is the GROWTREES algorithm described in Chapter 5. The current version of the TREEREL algorithm (Chapter 5) assumes that all the vertices in the PTS belong to the default case, i.e. all of their IPCM elements are unity. Until such time as the TREEREL algorithm is extended to handle this constraint, the PTSs may have no vertex which has an explicit IPCM. This implies the addition of one more test to Algorithm GROWTREES in Chapter 5. Thus the GROWTREES algorithm does not, at present, include any vertex into a PTS when it has an explicit IPCM. Whereas the PTS may ordinarily have "grown" past that vertex, the tree will now stop short of such a vertex. The introduction of this constraint, consequently, will force all vertices with explicit IPCMs to be left as part of the Kernel. The resulting PTSs may be smaller. The Compositions Table will not change much but the DoCore function will have more work to do since the Kernel will contain more vertices than otherwise. Note that this implementation restriction on the GROWTREES algorithm can be used at present in an *ad hoc* fashion to force particular components to be in the Kernel. To do this it is sufficient to assign to the component an IPCM all of whose elements are unity. The only effect of this currently is to force that component to be considered as part of the Kernel (see Section 7.3.1). The computation of path reliabilities in the Kernel however will not be affected.

6.9.2 Intra Component-Type Communication

6.9.2.1 Need for constraint

The second side-constraint deals with communication between components of like type. A minimal critical resource set (MCRS) of components may be composed of various different types of components. In the context of PMS structures some types of components are typically active and originate control information in the structure. Examples of active components are processors and direct-memory-access device controllers and other "smart" controllers. The remaining types of components are passive and accept control and commands from the active components. Examples of passive components are memories and input-output transducers. It is largely the case in typical PMS structures that active components will exchange information amongst each other, or with passive components while controlling them, or both. Thus, in general, paths for information flow will need to exist between active components and passive components in the structure and amongst the active components themselves. On the other hand paths need not be sought directly between passive components. Thus, including the probability of existence of K-edges between *passive* critical components during the Overlord main loop computations would lead to a pessimistic system reliability estimate. This is because components along those paths would be required to be functional which in reality are superfluous since the paths are never used for communication between the passive components. This is not always true since an active component might lie along one of the paths between the two passive components. However, this in turn implies that the path probability would be considered during some other iteration of the main loop when paths are being sought between the intermediate active component on that path and each of the passive components at either end of the path. The general constraint can be phrased as the directive "Do not attempt to account for K-edges between passive components of an MCRS".

6.9.2.2 Implementation

A study of the common types of PMS structures by the author seems to indicate that a weaker constraint might suffice. This weaker constraint, which was implemented, subsumes the more specific one above and can be phrased as follows: "Do not attempt to account for K-edges between components of <u>like</u> type except when otherwise explicitly specified by the user". Thus, for instance, paths should not be found between memories in one PTS and memories in another PTS through the Kernel. There were two reasons for requiring the user of ADVISER to specify generic component types, whose members <u>do</u> communicate amongst themselves in the operation of the PMS structure. The first is that the underlying graph model,

though it allows labels for vertices in the PMS graph, attaches no significance to these labels. Thus the user must specifically identify those component types whose members are active. The second reason is that in a typical PMS structure the majority of component types are passive thus making it easier to identify active component types with less effort. The model, therefore, assumes that communication between unlike component types is routine, that members of any passive component type do not communicate with each other, and that all component types are treated as though they were passive unless otherwise specified. In short the implemented constraint may be phrased: " Account for K-edges only between critical components of unlike type in the PMS structure, and not between critical components of like type *unless specified by the user.*"

6.9.2.3 Effect of constraint on algorithms

The implementation of the Intra Component-Type communication constraint affects the Kernel algorithms in two ways. First affected is the manner in which the PATHREL algorithm is used in the initial path generation phase when path-CRPs are hash-coded away for later use. The effect is in the way path-CRP computations are carried out for critical components in the Kernel. Path-CRPs are generated with the PATHREL algorithm for a pair of critical components in the Kernel if and only if they are of unlike types. Thus later, during the Overlord main loop iterations, when in the DoCore function the path-CRPs are fetched for some Compositions Table state, no CRPs will be found to exist for pairs of components of like type. Thus the DoCore function will assume that no paths exist between such components. which is, of course, the desired effect.

The second effect is in the way the DoCore function attempts to find paths from interface vertices to other interface and/or critical components in a given CCKS. In each state of the Compositions Table, for each PTS a set of component types is computed. This set specifies what various distinct types of critical components have been chosen from the PTS for the current Compositions Table state. Thus for each interface vertex it is always known which set of component types within its PTS are exchanging information with the critical components in the Kernel and in other PTSs. There are two cases in which it would be superfluous to find a path from that interface vertex to another vertex in the CCKS, namely:

1. Components of exactly one component type are currently active in the PTS of this interface vertex. It is proposed to find a path from the interface vertex to a critical component which is currently active in the Kernel. *The path is superfluous, if the component type of that critical component is the same as the single component type which is currently chosen in that PTS.* Note component types explicitly indicated by the user are exempt from this check.

2. Components of exactly one component type are currently chosen in the PTS of this interface vertex. It is proposed to find a path from this interface vertex to the interface vertex of a second PTS. The second PTS also has currently chosen components of exactly one type. *The path is superfluous if these two currently chosen types in the two PTSs are identical.* In other words according to the constraints no paths must be considered between vertices which act as channels between critical components of the same type. Note: Again, component types explicitly indicated by the user are exempt from this check.

In all other cases, i.e. if components of more than one type are currently chosen in a PTS, then there will be at least one critical component in the PTS and another in the Kernel, or another PTS, which are of unlike type and thus the paths from the interface vertex of this PTS to those other vertices are meaningful and must be considered for their reliability contribution.

6.9.3 Bounded Clustering of Critical Components

6.9.3.1 Need for constraint

The reliability computation for PMS structures differs in yet another way from the classical network reliability computation. We have seen that a major difference is that vertices of the PMS interconnection graph are not homogeneous and are classified naturally according to the distinct types of components present in the system. In addition to the Communication Axiom, the PMS system reliability is predicated upon a minimum number of pivotal components, termed critical components, being functional in the structure. Since there are a variety of component types represented in the structure, this stipulation on the minimum number of functional critical components requires to be strengthened to account for component types. We thus arrive at the minimal requirements which stipulate non-zero lower bounds on the number of components of each critical component type, which must be functional as a precondition to system success.

There is, however, a more subtle issue to be considered; one which forms the subject of this section and the reason for this third side-constraint. In many cases of PMS structures, a simple lower bound on the overall number of critical components of a particular type being functional, provides insufficient information for reliability computation. It is necessary in these cases to account in addition for the phenomenon that components of different types may be interdependent on each other in some facet of their operation. For instance, if some number of functional components of one such type occur in a particular substructure of the system, then it may be essential that at least a certain other number of components of an interdependent component type also be functional in the same substructure of the system to

achieve system success. Thus in a functional system of this type, clusters of functional critical components belonging to these interdependent types will be observed in the various substructures of the system. Furthermore, there will usually be a lower bound on the number of components of each critical component type in the cluster. We term this phenomenon Bounded Clustering of Critical Components.

As an example of this phenomenon consider a multiprocessor system composed of say eight processor buses, each with two processors and, say, eight local memory cards, and a bus-arbiter, amongst other components. Assume that these buses are connected together in some fashion (which is not of importance at the moment) so that it is always possible to satisfy the Communication Axiom if the minimum number of functional processors, memories and bus-arbiters are available. There are then a total of 16 processors, 64 memories and 8 bus-arbiters. Now say that a minimum requirement is

$$\psi(4,\text{Processor}) \wedge \psi(8,\text{LocalMemory}) \wedge \psi(4,\text{BusArbiter}) \qquad (6.3)$$

This overall stipulation allows too much latitude. We do not, for instance, consider a processor bus to be functional unless its bus-arbiter is functional. Additionally it may be necessary to have, say, two functional local memories *per* functioning processor on the processor bus. Therefore, it is useless to consider a system state wherein four processors are functional, two each on buses A and B, say, eight memories functional on a third bus C, and four bus-arbiters functional, one each on four other buses D, E, F and G. This would clearly not be a system success state although it satisfies the overall requirement expression (6.3) above.

On the contrary it is to be noted that a processor bus will be considered functional only if its bus-arbiter is functional and at least one of the two processors and two of the eight local memories on that bus are also functional. This represents a bounded clustering, in the processor-bus substructure of the system, of the critical component types BusArbiter, Processor and Memory. The lower bounds on the number of functioning components of each of these three types in the cluster are conveyed by the following set of inequalities:

$$\text{Number of BusArbiter} \geq 1$$

$$\text{Number of Processor} \geq 1$$

$$\text{Number of Memory} \geq 2 * \text{Number of Processor}$$

In general an arbitrary set of such inequalities may be specified which constrain the number of functional components of the various critical component types which are specified in the bounded-cluster constraint. Thus when the requirement integers in expression (6.3) above are fragmented over the various processor buses, these inequalities must be kept in mind for each processor bus.

The bounded clustering constraint, therefore, seeks to allow the user of ADVISER to specify which critical component types will cluster in the system and what the inequalities are which effectively place lower bounds on each critical component type in a cluster. Note that there may be several different kinds of clusters each with its own subset of the set of critical component types in the structure.

Two difficulties in handling the inequalities are to be noted here. First, if general sets of inequalities are allowed then the program will have to check before it even begins computation that these inequalities do have solutions. In other words an integer programming problem has to be solved for each set of inequalities. Of course, a brute force approach could be employed wherein all feasible states of the Compositions Table are checked against the inequalities. If none satisfy the constraints then the constraints are unsatisfiable for the given problem.

The second difficulty is arriving at an interpretation of the possibility that two user-specified cluster constraints address non-disjoint sets of critical component types. The author's current thinking is that if such intersecting cluster constraints are specified then a compound cluster constraint should be considered instead which has respectively the union of the intersecting sets of component types and the union of the sets of inequalities.

Note: *At present the ADVISER program implements only a weaker version of the cluster constraint.* Instead of allowing inequalities in their full generality it allows only a lower bound to be specified on the number of functioning components chosen from one of the cluster types. It does not allow the relating of numbers of chosen components of different types. Thus it is not currently possible to specify an inequality of the form

$$\text{Number of Memory} \geq 2 * \text{Number of Processor}$$

This deficiency had an effect during the experiment on ADVISER with the PLURIBUS architecture which is described in Chapter 7.

6.9.3.2 Effect on Algorithms

The bounded clustering constraint in the ADVISER framework is applied to the substructures of the PMS graph which are represented by the PTSs and the Kernel. The bounded clustering constraint directly affects the main loop of the Overlord routine. In the absence of this constraint, the process of generating the next feasible state of the Compositions Table produced a candidate feasible state for evaluation by the DoCore routine when the set of compositions in the rows of the Table satisfied the upper bounds placed on them by the resources available in each segment of the PMS graph. In the case of the bounded clustering constraint the inequalities effectively impose lower bounds as well. What were previously feasible states of the Table may no longer be so.

The constraint is implemented simply by examining each candidate feasible state of the Compositions Table as it is generated and before it is passed to the DoCore routine for the Communication Axiom test. The inequalities are tested for satisfaction against the values of cells in each *column* in the Table for the candidate Table state. Recall that if components of one of the types specified in a cluster constraint are chosen from a given segment, then components belonging to *all* of the other specified types must also be chosen from that segment, subject of course to the inequalities. Each column of the Compositions Table represents one segment of the PMS graph. Each column in the candidate feasible state of the Table is examined to discover whether the given cluster constraint applies to it. In other words the constraint applies to the column if one or more of the non-zero cells in the column represents one of the component types specified in the constraint. If the constraint applies to the column and the non-zero cell values in the column do not satisfy the constraint inequalities then the candidate is discarded and the next feasible state is generated.

There is one subtlety in these tests which must be kept in mind and is best explained through an example. Suppose that some cluster constraint specified by the user of ADVISER contains the following inequality

$$\text{Number of Processor} \geq 2 \qquad (6.4)$$

Assume that this constraint is applicable to a segment which has more than two critical components of type Processor. In other words the segment is able to satisfy the inequality (6.4). Now assume that one of the fragment atomic requirements imposed on the segment, during the fragmenting of the overall requirement on the PMS graph, is ψ(1 Processor) In other words the cluster constraint is stronger than the atomic requirement "at least 1 of Processor" which will also be satisfied by two or more functioning processors. However, if

cell for the component type "Processor", within the column corresponding to the given segment, had a one in it as usual, the candidate feasible state would fail the inequality test of (6.4) when it should not. Hence in such cases it is necessary to temporarily strengthen the fragment requirement to the minimum required by the inequality; in this case we replace ψ(1,Processor) by ψ(2,Processor) until the next fragmentation of the overall requirements occurs.

As noted above, the current implementation of ADVISER allows only concrete lower bounds (as opposed to being relative to other components types) to be placed in the cluster constraint. In addition, the current implementation specifically forbids multiple cluster constraints to have overlapping component type sets.

6.10 Simplification of Canonical Reliability Polynomials

In this section we discuss the simplification of the symbolic system reliability function which has been computed by ADVISER, prior to its being printed out. We emphasize here that the kinds of simplification envisioned are rudimentary and have none of the sophistication manifested by automatic symbol manipulation programs such as MACSYMA [Macsyma 77]. Such sophistication in algebraic manipulation was never considered to be a goal of this work.

Thus far all the algorithms described have used Canonical Reliability Polynomials (CRPs) as fundamental units upon which to operate. Therefore, at the conclusion of the main loop execution in the Overlord routine, the computed system reliability function is also in CRP form. We briefly review the nature of a CRP. Each unique component in the system is considered to exhibit failure behavior which is stochastically independent from that of other components in the system. A unique symbol is allocated to each component to represent its reliability. Each such component reliability symbol is associated with a unique bit position in a main bit vector (NORMVEC of Chapter 3). Likewise, during the generation of partial result CRPs for the PTSs, each partial result is also accorded a unique symbol which is associated with a unique bit position in an auxiliary bit vector (AUXVEC of Chapter 3). The assigning of unique bit positions for partial result CRPs stemmed from the need to reduce the number of expensive PMERGE and SMERGE operations which are performed. The partial results become factors of CRP terms through the presence of the unique bits assigned to them, thus allowing PMERGE and SMERGE operations on them to be postponed to a later time. This section discusses the process whereby the partial result CRPs associated with the nonzero bits in each AUXVEC are retrieved from hash tables and back-substituted into the CRP which

represents the system reliability function. The section ends with a description of the way in which the simplified system CRP is printed out.

The reader will recall from Chapter 3 that the juxtaposition of two factors in a CRP term represents the SMERGE of the probabilities represented by the factors. In the case of component probabilities, which are stochastically independent from each other by our assumptions, this SMERGE degenerates to a simple multiplication of those individual probabilities. However, as we have seen earlier, the probabilities represented by a pair of partial result CRPs can be interrelated if the subsets of the system components referenced by them are not disjoint. Then the SMERGE operation must be carried out on the pair of partial result CRPs to obtain the correct result. The method of generating these partial result CRPs guarantees that *two CRPs will represent dependent probabilities only if they are partial results for the same segment of the PMS graph*. Use is made of this fact during the simplification process. There are two important points to be made at this juncture, namely:

1. The SMERGE simplification will take place to a recursive depth of only one level. This is because the partial result CRPs themselves are devoid of AUXVECs in their terms.

2. The partial results for the PTS segments are all assigned bits in the AUXVECs of system CRP terms. Thus any NORMVEC bits in any term of the system CRP will refer to *only* the reliabilities of components in the Kernel.[34] Hence during the SMERGEing of CRPs represented by the AUXVEC bits of a term in the system CRP, the NORMVEC bits do not come into play since they are guaranteed to represent probabilities which are independent from those of the PTS partial results.

Due to the above observations, the NORMVEC and AUXVEC bit vectors of any given CRP term may be treated independently. The results may then be simply multiplied. In the following paragraphs the treatment of the AUXVEC and NORMVEC bit vectors are described separately for a typical term which has both a NORMVEC and and AUXVEC bit vector. The goal is to do any SMERGEs which are indicated thus leaving only those juxtapositions of bits which denote multiplication by virtue of their represented CRPs being independent probabilities. Finally the remaining multiplications are converted to exponentiations wherever two juxtaposed bits represent CRPs, which though representing independent probabilities are similar in form (i.e. share the same template) and, therefore, numerically evaluate to equal quantities. After this final simplification step the symbolic reliability function is printed out.

---

[34] *Excluding interface vertices*. The reliabilities of these are accounted for in the computation of the partial result CRPs for the PTSs. See Section 6.5.

6.10.1 NORMVEC processing

The bits in any NORMVEC bit vector are known to represent individual component probabilities which, by assumption, are stochastically independent. Thus all juxtapositions of 1-bits in the NORMVEC simply denote multiplication of the appropriate probabilities. In view of this, all pairs of bits in the NORMVEC are compared. If two bits represent components of the same type then their reliability functions are identical[35] and, therefore, the symbol for reliability of the type to which both the components belong, is raised to the power of two. Every succeeding bit which represents another component of the same type simply causes this power to be incremented by one. At the end of this processing, the simplified NORMVEC will consist of a set of factors each of which is a symbol for a component type reliability raised to a power. The power of a factor is simply the number of bits in the vector which represented a component of the same type as the factor represents. It is now obvious why part of the description of each component type, which was input by the user at the beginning of the program run, consisted of a "print-name" for the component type. These print names are the factor symbols referred to in this paragraph.

6.10.2 AUXVEC processing

During the simplification of the AUXVECs, for each pair of bits set to one in the AUXVEC of the term, the bits in the pair are compared on the basis of the partial result CRPs which they represent. If the CRPs were derived from different segments of the PMS graph then the sets of components they reference will be disjoint. Thus identical numerical results will be obtained if they are

- SMERGEd and the resulting CRP is numerically evaluated, or
- the individual CRPs are numerically evaluated and the resulting numbers multiplied.

Hence, in such cases, the SMERGE is not performed and the bits are left undisturbed. If the two bits being compared represent CRPs derived for the same segment of the PMS graph then the SMERGE is performed to give a third CRP (devoid of AUXVECs) which is then added to the Partial Results Hash Table and assigned a unique symbol of its own. It is also labelled with the identities of the two "parent" CRPs which were SMERGEd to form it. This is done since the same situation may occur in the simplification of another term in the CRP and the

---

[35] As noted in Chapter 2 it would be preferable to allow a pair of components to be classified as belonging to the *same* type while having *different* reliability functions. This is a simple matter of changing the functions in ADVISER which accept the problem description, and the change to the NORMVEC processing is obvious and trivial.

cost of the SMERGE may be avoided if the result was computed earlier and can be found in the hash table.

For each pair of AUXVEC bits compared there will be eight possible cases to be considered based on the hash keys of the partial result CRPs represented by the bits. Recall from Section 6.5.1.1 that a partial result CRP for a given PTS was uniquely identified by three attributes namely

(i) The PTS segment of the Neighbors Class Graph G′ which represents possibly several symmetric PTSs of the PMS graph G of which the given PTS is a member

(ii) The root vertex of the given PTS of G which distinguishes it from its symmetric "brothers", and

(iii) The atomic requirement for which the given CRP was derived.

The last item actually identifies a template in the Templates Table, and so two partial result CRPs are considered to be the same identical CRP if their PTS segments in G are in the same PTS of G′ (set of symmetric PTSs); their PTS segments in G have the same root vertex; and they derive from the same template. The eight cases described below are based on equality checks on these three attributes of two CRPs being compared:

Case 0 **Different segment of G′; Different root vertex; Different template:**
These are two completely different CRPs. They, therefore, represent independent probabilities and we may algebraically multiply them.

Case 1 **Different segment of G′; Different root vertex; Same template**
This is an impossible case. A template CRP will refer to the reliabilities of components in one of a set of symmetric PTSs of G (see Section 6.5.1.2). Therefore, if two CRPs are to have the same template they must also at least be derived for the same segment of G′ (i.e. the same set of symmetric trees).

Case 2 **Different segment of G′; Same root vertex; Different template:**
This is an impossible case. Two PTSs cannot have the same root vertex and belong to different segments.

Case 3 **Different segment of G′; Same root vertex; Same template:**
Impossible for the same reason as Case 2.

Case 4 **Same segment of G′; Different root vertex; Different template:**
The CRPs in this case were derived for different PTSs in the same set of symmetric PTSs. *Thus, the CRPs represent independent probabilities since their referenced component sets are disjoint.* They may be directly multiplied in the simplified reliability function.

Case 5 **Same segment of G′; Different root vertex; Same template:**

The PTSs are two symmetric trees in the same set. The fact that the templates are the same implies that the CRPs compute the reliability of physically symmetric PTSs *under the same atomic requirement*. Thus, numerically, the two CRPs will evaluate to be equal. Consequently we may algebraically square the template to get the equivalent value. More precisely, the exponent count of the symbol for the template in the current term is incremented by one. This is the power to which the template is to be raised at the end of simplification of the given system CRP term.

Case 6 Same segment of G´; Same root vertex; Different template:
Both partial result CRPs in this case refer to the same PTS of G but were derived for different atomic requirements. *Therefore, they represent dependent reliabilities and must be SMERGEd*. The result of the SMERGE is entered into the Partial Results table with a new index.

Case 7 Same segment of G´; Same root vertex; Same template:
This is an "impossible" case. It implies that a given partial result was not assigned a unique bit in the AUXVEC

6.10.3 Final algebraic simplification

During the first pass over the system CRP the NORMVEC and AUXVEC simplifications are performed on each term in the CRP. The result of this first pass is that all SMERGEs which needed to be performed have been carried out and only algebraic multiplications remain. Repeated occurrences of juxtapositions in different CRP terms of the same two bits which require SMERGEing of their CRPs will cause the SMEGE to occur only once. Whereupon, the resultant CRP will be inserted into the hash table and appended with the identities of its "parent" CRPs. This enables avoidance of redundant SMERGE operations. Note that several terms in the simplified system CRP so far may reference the same set of templates. The numerical evaluation of the system CRP can take advantage of this fact by computing the numerical values of the template CRPs just once and using them repeatedly for each term which references the corresponding templates. *In software terms, program statements may be generated which evaluate each template CRP and store the result in a temporary variable which bears as its name the unique symbol of the CRP.*

A final $O(N^2/2)$ pass is made over the simplified system CRP thus far to do algebraic simplification. This consists in comparing each term in the system CRP to each of its successors in the list of terms and, in case of equality, algebraically adding the signed coefficients. Where the addition leads to zero, the terms have cancelled and may be discarded. At the end of this simplification process the resulting list of terms represents the system reliability function and is ready for printing out.

At this point we also note that, preparatory to printing out the results, the template CRPs also need to be simplified before they can be output. Since all the template CRPs generated during the initial phase of partial result generation have only NORMVECs, the simplification of their terms is carried out in the manner of Section 6.10.1. Finally, they too undergo algebraic simplification as above.

6.11 Printing of Results

At the present time, the ADVISER program is able to print out the computed symbolic system reliability function *as the text of a program module which computes the function*. Currently the program text may be in either FORTRAN or SAIL [Reiser 76]. The module may then be compiled and loaded along with other software which may make use of it to draw plots, or for other numeric computation. An third output mode causes ADVISER to print the function in a simple expression syntax, unencumbered by programming constructs, which will make it suitable as input to the symbol manipulation system MACSYMA [Macsyma 77] (see Chapter 7 for examples of how the MACSYMA option is used).

Figures 6-15 and 6-16 show the output for a simple reliability function in FORTRAN and SAIL respectively.[36] The component type definitions, PMS structure definitions and the requirements expression have been output as comments preceding the program statements. Each program module consists of a sub-program or procedure whose name may be supplied by *the user but defaults to RSYS if not supplied*. The program, since it computes $R_{sys}(t)$, takes a parameter T which is the time at which the system reliability is to be determined. A single floating point value, $R_{sys}(T)$, is returned by the procedure. Another feature, which will be noticed in these programs produced by ADVISER, is that variables are declared, one for each component type, bearing as their names the print names declared for the respective component type (see Table 2-1 in Chapter 2). Each variable is initialized to the computed reliability of a component of the type represented by the variable, at time T specified as a parameter to the program. These values are then used in the computation of the temporary variables and the main reliability function.

The temporary variables introduced into the SAIL program (Figure 6-16) are of the form *!T!n* where *n* is an integer. Likewise, in the FORTRAN version (Figure 6-15) the temporaries are named *XXXm* where *m* is also an integer. As was described earlier, all the simplified CRPs

---

[36] As an aid to reading the FORTRAN version, we mention here that all continuation lines are prefixed in column 6 with a "$" sign.

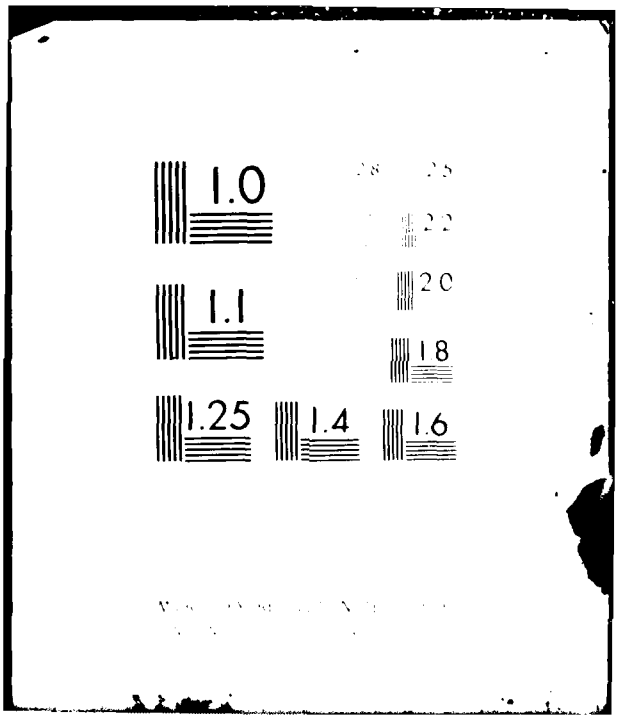
which were referred to in the various terms of the system CRP, via the Partial Results Hash Table, are computed first. Their values are assigned to the unique temporaries as may be seen in the examples. These values are then used wherever required in the system CRP. A question might arise as to whether the final simplified system reliability function would have less terms if the templates were back-substituted into it and algebraic simplification were carried out. This would be true in the case of completely symmetric PMS structures which would yield algebraically simple, factored reliability functions. However, any slight asymmetry will cause the final reliability function to be less easily factorable, or not at all. Thus, in most cases, this "factorization" based on templates would seem to be at least somewhat beneficial from the standpoint of numeric computation. If error magnitudes during numerical reliability computation are a serious issue then the symbolic function may be factored using a symbol manipulation program such as MACSYMA before the numeric computation is performed.

```
C-------------------------------------------------------------------------
C ** FORTRAN Module for Reliability Function evaluation
C **      produced by ADVISER on Sunday, 25 Jan 81 at 22:09:45 for [4,1367]
C-------------------------------------------------------------------------
C ** Task Title: EXAMP.PMS -- Running example in thesis
C
C ** Requirements on the Structure were:
C
C       (1-OF-CPU AND 1-OF-MPR AND 1-OF-MSH AND 2-OF-DSK)
C
C ** Component-Type definitions for this task:
C
C  INDEX  TYPENAME     PRINTNAME    REL.FN.   PARAMS
C  -----  --------     ---------    -------   ------
C      0  M.SHARED     MSH          Expon.    Lambda= .00100000
C      1  CPU          CPU          Expon.    Lambda= .00200000
C      2  BUS          BUS          Weibull   Lambda= .00010000
C                                               Alpha= .90000001
C      3  LINK         LNK          Weibull   Lambda= .00100000
C                                               Alpha= .90000001
C      4  M.PRIMARY    MPR          Expon.    Lambda= .00100000
C      5  DISK         DSK          Weibull   Lambda=10.00000000
C                                               Alpha= .93000001
C      6  K.DISK       KDK          Weibull   Lambda=6.00000000
C                                               Alpha= .89000001
```

AD-A112 713 CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER --ETC F/G 9/2
AUTOMATIC GENERATION OF RELIABILITY FUNCTIONS FOR PROCESSOR-MEM--ETC(U)
FEB 81 V KINI N00014-77-C-0103
UNCLASSIFIED CMU-CS-81-121 NL

3 OF 3
AD A

END
DATE FILMED
4-82
DTIC


1.0
1.1
1.25
1.4
1.6
2.0
1.8

```
C ** PMS Structure Definitions for this  task:
C
C  INDEX  NAME         TYPE         NNEIG  NEIGHBORS
C  -----  ----         ----         -----  ---------
C      0  MSH.1        M.SHARED         2  (P.1, P.2)
C      1  P.1          CPU              2  (MSH.1, S.1)
C      2  P.2          CPU              2  (MSH.1, S.2)
C      3  S.1          BUS              4  (P.1, L.1, MP.1, K.1)
C      4  S.2          BUS              4  (P.2, L.1, MP.2, K.2)
C      5  L.1          LINK             2  (S.1, S.2)
C      6  MP.1         M.PRIMARY        1  (S.1)
C      7  MP.2         M.PRIMARY        1  (S.2)
C      8  K.1          K.DISK           3  (S.1, D.1, D.2)
C      9  K.2          K.DISK           3  (S.2, D.3, D.4)
C     10  D.1          DISK             1  (K.1)
C     11  D.2          DISK             1  (K.1)
C     12  D.3          DISK             1  (K.2)
C     13  D.4          DISK             1  (K.2)

C

C-------------------------------------------------------------------------
C
C ***  Begin Reliability Function evaluation code:


      REAL FUNCTION RSYS (T);
      IMPLICIT REAL (A-Z)

      WEIBUL(LAMBDA,ALPHA,TIME)=EXP(-(LAMBDA*1E-6*TIME)**ALPHA)

      MSH = EXP(-0.001000 * 1E-6 * T)
      CPU = EXP(-0.002000 * 1E-6 * T)
      BUS = WEIBUL( 0.000100 , 0.900000 , T )
      LNK = WEIBUL( 0.001000 , 0.900000 , T )
      MPR = EXP(-0.001000 * 1E-6 * T)
      DSK = WEIBUL( 10.000000 , 0.930000 , T )
      KDK = WEIBUL( 6.000000 , 0.890000 , T )
C ** End of expressions for calculating individual reliabilities:

      XXX0 = BUS * MPR

      XXX1 = BUS * DSK**2 * KDK

      XXX2 = 2.0 * BUS * DSK * KDK  -  BUS * DSK**2 * KDK

      XXX3 = BUS * MPR * DSK**2 * KDK

      XXX4 = 2.0 * BUS * MPR * DSK * KDK  -  BUS * MPR * DSK**2 *
     $KDK

C ** End of template evaluating expressions;

      MODREL = 0

      MODREL = 2.0 * MSH * CPU * XXX3  -  MSH * CPU**2 * XXX3**2  +
     $2.0 * MSH * CPU**2 * LNK * XXX0 * XXX1  -  2.0 * MSH * CPU**2 *
     $LNK * XXX3 * XXX0  +  2.0 * MSH * CPU**2 * LNK * XXX4 * XXX2
     $  -  2.0 * MSH * CPU**2 * LNK * XXX3 * XXX2  -  2.0 * MSH *
     $CPU**2 * LNK * XXX4 * XXX1  -  MSH * CPU**2 * LNK * XXX4**2  +
     $2.0 * MSH * CPU**2 * LNK * XXX4 * XXX3  +  MSH * CPU**2 * LNK *
     $XXX3**2
C **  End of System Reliability computation;
```

```
      RSYS = MODREL
      RETURN
      END
```

Figure 6-15: An example of a computed reliability function printed in FORTRAN

```
COMMENT
------------------------------------------------------------------------
SAIL Module for Reliability Function evaluation
   produced by ADVISER on Sunday, 25 Jan 81 at 22:10:16 for [4,1367]
------------------------------------------------------------------------
Task Title: EXAMP.PMS -- Running example in thesis


Requirements on the Structure were:

(1-OF-CPU AND 1-OF-MPR AND 1-OF-MSH AND 2-OF-DSK)

Component-Type definitions for this task:

INDEX  TYPENAME     PRINTNAME   REL.FN.   PARAMS
-----  --------     ---------   -------   ------
    0  M.SHARED     MSH         Expon.    Lambda= .00100000
    1  CPU          CPU         Expon.    Lambda= .00200000
    2  BUS          BUS         Weibull   Lambda= .00010000
                                          Alpha= .90000001
    3  LINK         LNK         Weibull   Lambda= .00100000
                                          Alpha= .90000001
    4  M.PRIMARY    MPR         Expon.    Lambda= .00100000
    5  DISK         DSK         Weibull   Lambda=10.00000000
                                          Alpha= .93000001
    6  K.DISK       KDK         Weibull   Lambda=6.00000000
                                          Alpha= .89000001

PMS Structure Definitions for this  task:

INDEX  NAME         TYPE         NNEIG  NEIGHBORS
-----  ----         ----         -----  ---------
    0  MSH.1        M.SHARED         2  (P.1, P.2)
    1  P.1          CPU              2  (MSH.1, S.1)
    2  P.2          CPU              2  (MSH.1, S.2)
    3  S.1          BUS              4  (P.1, L.1, MP.1, K.1)
    4  S.2          BUS              4  (P.2, L.1, MP.2, K.2)
    5  L.1          LINK             2  (S.1, S.2)
    6  MP.1         M.PRIMARY        1  (S.1)
    7  MP.2         M.PRIMARY        1  (S.2)
    8  K.1          K.DISK           3  (S.1, D.1, D.2)
    9  K.2          K.DISK           3  (S.2, D.3, D.4)
   10  D.1          DISK             1  (K.1)
   11  D.2          DISK             1  (K.1)
   12  D.3          DISK             1  (K.2)
   13  D.4          DISK             1  (K.2)


------------------------------------------------------------------------
;
COMMENT Begin Reliability function evaluation code;

ENTRY RSYS;

BEGIN "Declaration!Block"
```

```
INTERNAL SIMPLE REAL PROCEDURE RSYS (REAL T);
BEGIN "Calculation!of!RSYS"

REAL !System!Reliability;
REQUIRE "{}<>" DELIMITERS;
DEFINE WEIBULL(LAMBDA,ALPHA) = {EXP( - (LAMBDA * 1@-6 * T)^ALPHA)},

REAL
        MSH,CPU,BUS,LNK,MPR,DSK,KDK
;       COMMENT End of individual Reliability Function variable declarations,


REAL
        !T!0,!T!1,!T!2,!T!3,!T!4
;       COMMENT End of template variable declarations,

MSH ← EXP(-0.001000 * 1@-6 * T);
CPU ← EXP(-0.002000 * 1@-6 * T);
BUS ← WEIBULL( 0.000100 , 0.900000 );
LNK ← WEIBULL( 0.001000 , 0.900000 );
MPR ← EXP(-0.001000 * 1@-6 * T);
DSK ← WEIBULL( 10.000000 , 0.930000 );
KDK ← WEIBULL( 6.000000 , 0.890000 );

        COMMENT End of expressions for calculating individual reliabilities,

!T!0←
        BUS * MPR
        ;
!T!1←
      * BUS * DSK^2 * KDK
        ;
!T!2←
        2.0 * BUS * DSK * KDK  -  BUS * DSK^2 * KDK
        ;
!T!3←
        BUS * MPR * DSK^2 * KDK
        ;
!T!4←
        2.0 * BUS * MPR * DSK * KDK  -  BUS * MPR * DSK^2 * KDK
        ;
        COMMENT End of template evaluating expressions;

!System!Reliability←0;


!System!Reliability←
        2.0 * MSH * CPU * !T!3  -  MSH * CPU^2 * !T!3^2  +  2.0 *
        MSH * CPU^2 * LNK * !T!0 * !T!1  -  2.0 * MSH * CPU^2 * LNK *
        !T!3 * !T!0  +  2.0 * MSH * CPU^2 * LNK * !T!4 * !T!2  -  2.0
         * MSH * CPU^2 * LNK * !T!3 * !T!2  -  2.0 * MSH * CPU^2 *
        LNK * !T!4 * !T!1  -  MSH * CPU^2 * LNK * !T!4^2  +  2.0 *
        MSH * CPU^2 * LNK * !T!4 * !T!3  +  MSH * CPU^2 * LNK * !T!3^2
        ;       COMMENT End of System Reliability computa ion


RETURN (!System!Reliability);

END "Calculation!of!RSYS";

END "Declaration!Block";
```

Figure 6-16: An example of a computed reliability function printed in SAIL

6.12 Summary

This chapter has provided an overall view of the ADVISER program while giving specific details as regards the Overlord routine within it. The Overlord routine controls and synthesizes the efforts of subordinate functions in the program.

The initial task is to input the problem specifications which, in order, consists of the PMS component-type definitions, the PMS interconnection graph, the boolean requirements expression, and side-constraints, if any. The PMS interconnection graph is then analyzed for symmetries which might help in reducing the total amount of computation necessary. Any pendant tree subgraphs of the PMS graph are then isolated and form the known-segments apart from the Kernel. The basis for such a segmenting of the graphs is that special reliability computation techniques are known for known-segments but only simple pathfinding methods are applied to the Kernel.

The next step analyzes the boolean requirements expression to determine what atomic requirements will ever be applied, during the course of the computation, to any given known-segment or the Kernel. Partial results are computed for each of these known-segments for each atomic requirement imposed. The partial results are then hash coded for quick recovery during the several occasions in which each partial result is expected to be used. Any symmetry in the PMS graphs is exploited here by computing partial results for one of a set of symmetric subgraphs and extending them to the rest by storing the results as templates.

Requirements expressions containing disjunctions are converted to a disjunction-of-conjunctions, or sum-of-products form and only pure conjunctive requirements are passed to the Overlord routine main loop. The partial result CRPs resulting from these pure conjunctive requirements are finally PMERGEd to provide the system CRP.

The Overlord routine then enters its main loop for each purely conjunctive requirement passed to it. Here cases are generated of instances where the functional required components are scattered in various ways throughout the structure. Each case is checked to see if its particular scattering of components satisfies the Communication Axiom and the side-constraints (if any). The computation of the reliability contribution of the Kernel is crucial in this determination and is deputed to the function DoCore. Each case which satisfies the Communication Axiom represents a subset of system success states. The contribution of such cases is accounted for in disjunction by using the PMERGE algorithm, since any one of them may provide a functional system.

The canonical reliability polynomial for the system, resulting from the PMERGE of the CRPs returned by the Overlord routine operating on the pure conjunctive requirements, is simplified by taking into account the fact that identical components have identical reliability functions. The ADVISER program finally prints out a program which computes the system reliability at a time T which is passed to the program as a parameter. This program may be compiled and loaded with other programs which desire to utilize the computed reliability function.

Chapter 7
Examples and Results

This chapter describes experiments with the ADVISER program which were used to validate the reliability functions produced by it, thereby raising confidence in its useability. The ADVISER program is written in the BLISS-10 language [Wulf 71] for the Digital Equipment Corp. PDP-10 architecture. The program occupies approximately 40K 36-bit words of memory and automatically expands to accommodate problem sizes.

7.1 Validation of ADVISER

In programming practice, programs of any reasonable size may usually be expected to contain errors when initially constructed. Usual methods of program testing include generating and using sets of test input data which will cause all paths of flow of control through the program to be exercised. If the input data and output results are such that they can be easily duplicated by hand or engender confidence of correctness upon simple examination then the testing process is easier and can be made very thorough. Testing is much harder in the case of a program such as ADVISER where the rationale for building the program is to compute reliability functions which would otherwise be too tedious, complicated and subject to error when produced manually. It is difficult in addition to cursorily examine a polynomial and pronounce it as being the correct system reliability polynomial for the given PMS structure under the given requirements. On the other hand, it is also the case that complex programs such as compilers are never fully debugged whereas user confidence in them grows with prolonged use and with experience as to which types of input data are likely to cause errors in the output and should be avoided. It would appear that short of doing a full scale formal verification of ADVISER its usefulness would have to be determined over a period of time during which intensive use would expose most major errors. Until such time as confidence in the program is sufficient its output would have to be viewed with at least mild suspicion.

Under these conditions it was decided that initial tests of ADVISER should proceed along two major avenues, namely

- Compute the result for a set of representative PMS structures, both using ADVISER and "by hand" (i.e. largely manually although assisted by machine in some ways so as to relieve tedium). Choose the structures so that they represent the major types of structures which are planned for in the program.

- Use the results of other efforts by independent researchers and compare them to the output of ADVISER applied to the same problem.

In Sections 7.2 and 7.3 we describe both kinds of tests which were conducted with ADVISER.

In addition to inspection of the output of ADVISER, at least one other form of check proved to be so useful during the debugging of the ADVISER program that it is now standardly performed for each CRP which is ever manipulated in the program. For any reliability function the following two properties hold true:

1. The setting of all the factors in all its terms to zero should cause the function to evaluate identically to zero.

2. Likewise, the setting of all the factors in all its terms to unity should cause the function to evaluate identically to unity.

In order for Property 1 to hold true the function must not have a constant term. This is already true of CRPs as ADVISER uses them and thus the first property uniformly holds true for CRPs. For Property 2 to hold true the sum of the coefficients of all the terms in the CRP must be 1. This is so since all the polynomials manipulated by ADVISER are in canonical form. This boundary-value test is simple to conduct and it is performed in ADVISER upon the result of each SMERGE or PMERGE operation and, consequently, on the final system reliability polynomial. This very simple "go no-go" expedient was instrumental in trapping many errors in the program during the various stages of its construction and testing.

7.2 Comparison to manual calculations

Three types of PMS structures were chosen for the test in which the output of ADVISER was compared to manually derived results. By manually derived results we mean here that the *analysis* of the PMS structure and its functional states was done manually in order to produce an intermediate representation (which is the usual current practice) and then a program was used to reduce the intermediate form to the final result. The output from ADVISER was then compared with this result. The intermediate representation in the manual case was also chosen to be the Series-Parallel Reliability Block Diagram. The program constructed as an aid to hand calculation to help solve the intermediate representation was written in the INTERLISP language [Teitelman 78] which is a variety of LISP. The outputs of ADVISER and the LISP program are both symbolic expressions and thus may be compared.

In most instances the comparison was too tedious to do by simple examination. This was true in particular because ADVISER introduces temporary variables which represent the intermediate results generated during the computation. Thus in order to be able to compare the two expressions, the symbolic values of the temporary variables have to be substituted back into the system reliability polynomial generated by ADVISER, and the latter algebraically simplified, before comparison can begin. The MACSYMA program for symbol manipulation [Macsyma 77] was an invaluable tool in this regard. Both ADVISER and the INTERLISP programs were constructed to output command files which would cause the results of their respective computations to be loaded as polynomials into MACSYMA. The latter could then be invoked to compare them.

The series of figures and program output listings on the following pages will show the results of three of the experiments. In these experiments three similar simple PMS architectures were chosen along with similar requirements expressions. The interconnection schemes of these examples presented ADVISER with three cases which are dealt with each quite differently in the program despite their superficial similarity. These three examples were also chosen for inclusion here in part to display the way in which the number of functional states of a structure, and consequently its reliability function, can change with a small change in the interconnection scheme of the PMS architecture. Only the results of using a single requirements expression per example are included here for purposes of exposition although other experiments were carried out with satisfactory results.

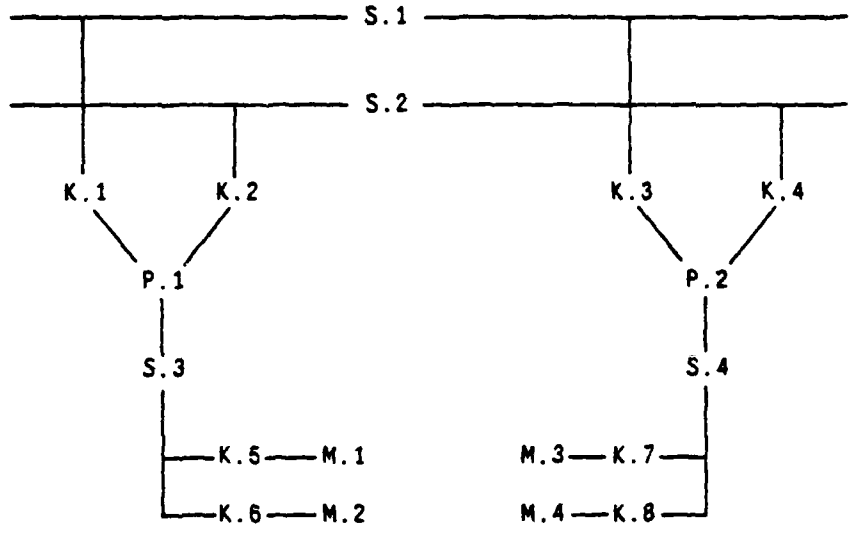
7.2.1 The DEC1.PMS example

We now describe the first of the three experiments in detail. The process for the other two experiments was very similar and for them only the results of the ADVISER run and the hand calculation are presented in Section 7.2.2.

Figure 7-1 shows the first of the three types of PMS structures chosen for manual evaluation. Two processors P.1 and P.2 can communicate through one or both of two interprocessor buses (S.1 and S.2) and each has its own local bus (S.3 and S.4) with two disk memories apiece (M.1 through M.4). The requirements chosen for this structure were "ψ(1,P) $\wedge$ ψ(2,M)". Figure 7-2 shows the manually derived series-parallel reliability block diagram for this set of givens.

The SPRBD of Figure 7-2 is explained as follows. Each path through the SPRBD from source to sink vertex describes one functional state of the PMS structure under the given

PMS Diagram:



Requirements:

$$\psi(1,P) \wedge \psi(2,M)$$

Figure 7-1: Example DEC1.PMS -- PMS Diagram and Requirements.

requirements, both of which are shown in Figure 7-1. Appropriate arcs of the SPRBD are numbered in Figure 7-2 so that the paths may be described. The single-arc path {1} indicates that one functional state is achieved when the components in the set {P.1,S.3,K.5,K.6,M.1,M.2} are functional. In other words the required one processor and two disk memories are provided by the P.1 processor bus. Likewise the single-arc path {2} indicates a functional state achieved when the requirements are met by he other processor bus under P.2. The parallel combination of the arcs {4} and {5} in Figure 7-1 describes the fact that *at least* one of the interprocessor buses (and the associated bus interfaces) needs to be functional. Likewise, the four arcs {7},{8},{9} and {10} in parallel describe the four possible ways of having two disks memories functional from the four that are available in the PMS structure. Thus, for instance, path {3,4,6,7} describes a functional state wherein the interprocessor bus S.1 is functional, P.1 and P.2 are both functional (thus satisfying the $\psi(1,P)$

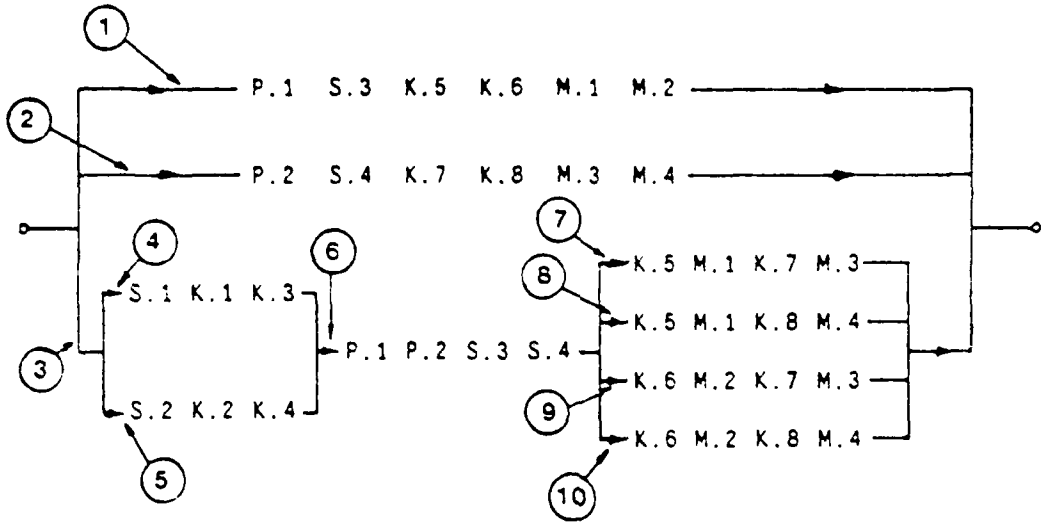


Figure 7-2: Example DEC1.PMS -- Hand-constructed SPRBD for given requirements.

atomic requirement) and the two functional disks are M.1 and M.3. Note that in all these four cases both processors need to be functional so that both functioning disks are accessible

The following listings of files show the process by which it was determined that the symbolic reliability function resulting as a solution of the SPRBD in Figure 7-2 was identical to that produced by ADVISER for the same problem. In all of these listings this first example is referred to as DEC1.PMS. Preceding each listing are comments to aid in its interpretation

Listing 1, Example DEC1.PMS

The following is a listing of the command file prepared for input to ADVISER in order to set up the example problem DEC1. Listing 2 for example DEC1 starting on Page 192 shows how it is used.

```
input title
DEC1.PMS -- A dual bus, 2 processor-bus architecture
```

```
input types
dynabus          dbus             E       0.0001
k.dbus           ks               E       6.000
pdp11            p                W       8.00     0.89
unibus           ubus             E       0.0001
disk             ms               W       10.000   0.91
k.disk           km               W       6.000    0.86

input pms
s.1              dynabus          k.1     k.3
s.2              dynabus          k.2     k.4
k.1              k.dbus           s.1     p.1
k.2              k.dbus           s.2     p.1
k.3              k.dbus           s.1     p.2
k.4              k.dbus           s.2     p.2
p.1              pdp11            k.1     k.2      s.3
p.2              pdp11            k.3     k.4      s.4
s.3              unibus           p.1     k.5      k.6
s.4              unibus           p.2     k.7      k.8
k.5              k.disk           s.3     ms.1
k.6              k.disk           s.3     ms.2
k.7              k.disk           s.4     ms.3
k.8              k.disk           s.4     ms.4
ms.1             disk             k.5
ms.2             disk             k.6
ms.3             disk             k.7
ms.4             disk             k.8

input restriction selftalk
pdp11

set watch run
```

Listing 2, Example DEC1.PMS

This listing shows the teletype session with ADVISER which solved the problem and printed out the solution. The characters typed in by the user for this problem are underlined. The "@dec1.pms" command causes the file shown in Listing 1 to be read and its lines executed as a series of commands. The double asterisk prompt characters of the program indicate that it is currently within a command file. The reading of the command file causes the PMS structure and the component types to be defined. The command "input restriction selftalk" implements the Intra Component-Type Communication side constraint discussed in Chapter 6. The command "set watch run" causes ADVISER to print run times of each command and each computation phase during the reliability calculation process. In the declaration of the component types the reliability function type "E" stands for the exponential distribution and the following real number is the failure rate for the distribution. The reliability function type "W" stands for the Weibull distribution and the following two real numbers are respectively

the scale and shape parameters of the distribution. This information is not used in the present example in which only symbolic manipulations are intended. The commands set up in the command file may just as well be entered individually during the teletype session but the command file option allows the user to think about and prepare an error free input offline with an editor. During the reliability function computation ADVISER prints messages indicating its progress through various phases. The "print" command is used with the "macsyma" option to obtain the computed symbolic function in a form suitable for input to MACSYMA. The resulting printout is shown in Listing 3 starting on Page 194.

```
@adviser
ADVISER 2A(6)   Wednesday 7 Jan 81   10:52:03
*@dec1 pms
**input title
Title: DEC1.PMS -- A dual bus, 2 processor-bus architecture
**
**input types
Input component types and associated print-names; end with blank line
Types   |       Print-names     | Rel.Fn.  | Lambda  | {Alpha}
-----------------------------------------------------------
dynabus         dbus            E       0.0001
k.dbus          ks              E       6.000
pdp11           p               W       6.00    0.69
unibus          ubus            E       0.0001
disk            ms              W       10.000  0.91
k.disk          km              W       6.000   0.86

**input pms
Input graph in format (end with blank line):
Component name | Typename | Neighbour,Neighbour.....
-----------------------------------------------------
s.1             dynabus         k.1     k.3
s.2             dynabus         k.2     k.4
k.1             k.dbus          s.1     p.1
k.2             k.dbus          s.2     p.1
k.3             k.dbus          s.1     p.2
k.4             k.dbus          s.2     p.2
p.1             pdp11           k.1     k.2     s.3
p.2             pdp11           k.3     k.4     s.4
s.3             unibus          p.1     k.5     k.6
s.4             unibus          p.2     k.7     k.8
k.5             k.disk          s.3     ms.1
k.6             k.disk          s.3     ms.2
k.7             k.disk          s.4     ms.3
k.8             k.disk          s.4     ms.4
ms.1            disk            k.5
ms.2            disk            k.6
ms.3            disk            k.7
ms.4            disk            k.8

**input restriction selftalk
Input list of Type names:
pdp11

**set watch run

[.00]
**
```

```
[.03]
*input requirements
Input boolean function (X of M AND/OR Y of M etc.):
1 of pdp11 and 2 of disk

[.12]
*get reliability
Generating symmetries.....
[.07]
Hashing Kernel term lists.....
[.06]
Hashing PTS term lists.....
[.05]
Setting up table space.....
Computing Reliability Function.....
CG: 1
######
        SC: 6
Collapsing CRPTree.....
######
[.23]
[.31]
Releasing table space.....
Reducing Reliability Function.....

Number of terms to be processed = 60. Here goes....
############################################################
Doing Algebraic Simplification....
###########
Terms remaining = 8
[.57]
Done!

[1.14]
*openo decl.mcs

[.10]
*print reliability macsyma

[.10]
*closo

[.04]
*exit

EXIT
@
```

Listing 3, Example DEC1.PMS

This listing shows the MACSYMA command file which was printed by ADVISER. In this file character strings which are meant to be comments to MACSYMA are bracketed (as in the PL/I language) by the delimiters "/*" and "*/". Semicolons are activating characters which tell MACSYMA that a command has been completely typed. A colon is the assignment

character which causes the symbolic value to its right to be assigned to the variable whose name appears on its left. The variables whose names begin with the characters "%%T" are temporary variables which are introduced by ADVISER to hold the intermediate result CRPs which were generated during the computation of the system reliability function. Listing 5 starting on Page 196 shows a teletype session with MACSYMA which uses the command file shown in this listing.

```
/*
-------------------------------------------------------------------------
MACSYMA Module for Reliability Function manipulation
   produced by ADVISER on Wednesday,  7 Jan 81 at 10:54:25 for [4,1367]
-------------------------------------------------------------------------
Task Title: DEC1.PMS -- A Tandem/16-like architecture, Version 1

Requirements on the Structure were:

(1-OF-P AND 2-OF-MS)
-------------------------------------------------------------------------
*/


%%T1:
        P * UBUS * MS↑2 * KM↑2;
%%T2:
        2 * P * UBUS * MS * KM  -  P * UBUS * MS↑2 * KM↑2;
/* End of temporary variable initializations */

System%Reliability: 0;
System%Reliability:
        2 * DBUS * KS↑2 * %%T2↑2  -  4 * DBUS * KS↑2 * %%T1 * %%T2
          +  2 * DBUS * KS↑2 * %%T1↑2  -  DBUS↑2 * KS↑4 * %%T2↑2  +  2
         * DBUS↑2 * KS↑4 * %%T1 * %%T2  -  DBUS↑2 * KS↑4 * %%T1↑2  +
        2 * %%T1  -  %%T1↑2
        ; /*End of System Reliability computation*/

FACTOR(%);
```

Listing 4, Example DEC1.PMS

This listing is the output from the INTERLISP program which was written to solve SPRBDs such as the one in Figure 7-2. The listing is in the format of a MACSYMA command file which sets the variable SYSREL to the symbolic expression which results from the solution of the hand-constructed SPRBD of Figure 7-2. The use of this command file is shown in Listing 5 below.

```
/* Reliability Function printed by LISP at 11-Jan-81 19:25:45 */
SYSREL:
+2*KM↑2*MS↑2*P*UBUS-1*KM↑4*MS↑4*P↑2*UBUS↑2+8*DBUS*KM↑2*KS↑2*MS↑2*P↑2*UBUS↑2
-16*DBUS*KM↑3*KS↑2*MS↑3*P↑2*UBUS↑2+8*DBUS*KM↑4*KS↑2*MS↑4*P↑2*UBUS↑2-4*DBUS↑2*KM
↑2*KS↑4*MS↑2*P↑2*UBUS↑2+8*DBUS↑2*KM↑3*KS↑4*MS↑3*P↑2*UBUS↑2
-4*DBUS↑2*KM↑4*KS↑4*MS↑4*P↑2*UBUS↑2;
```

Listing 5, Example DEC1.PMS

This listing shows the MACSYMA teletype session which confirms the equality of the symbolic expressions generated by ADVISER for the DEC1.PMS example and by the INTERLISP program from the hand-constructed SPRBD of Figure 7-2. The outputs from ADVISER and INTERLISP were shipped across the Arpanet to the host computer MIT-MC where the MACSYMA program resides. At MIT-MC the ADVISER output was kept in the file *KINI EGADV* and the results of the hand-calculation were kept in the file *KINI EGHND*. MACSYMA prompts for command lines with the characters "(Cn)" where n represents consecutive integers. This allows the user to refer to previously typed commands. The results of MACSYMA's computations are prefixed by the characters "(Dm)" where m also represents consecutive integers. Characters typed by the user during this session are underlined. The "batch" function in MACSYMA causes a command file to be read. The first command file read in was *KINI EGADV* and this set the variable "System%Reliability" to the symbolic expression computed by ADVISER with all the temporary variables substituted in and the result simplified (line (D5) in the listing below). The "FACTOR(%)" causes the symbolic expression on the immediately previous "D" line (in this case the value of System%Reliability) to be factored. The command file *KINI EGHND* was read in next with the "batch" function and caused the variable "SYSREL" to be set to the symbolic expression computed using INTERLISP. Finally, line (C9) requests MACSYMA to expand to its simplest terms the expression resulting from the subtraction of the symbolic values of SYSREL and System%Reliability. The result is zero indicating that the expressions are identical.

```
*:e

This is MACSYMA 293

FIX293 6 DSK MACSYM being loaded
Loading done

(C1) batch(kini.egadv);
```

```
(C2) /*
-----------------------------------------------------------------------
MACSYMA Module for Reliability Function manipulation
   produced by ADVISER on Wednesday,  7 Jan 81 at 10:54:25 for [4,136?]
-----------------------------------------------------------------------
Task Title: DEC1.PMS -- A Tandem/16-like architecture, Version 1


Requirements on the Structure were:

(1-OF-P AND 2-OF-MS)
-----------------------------------------------------------------------
*/


%%T1:
        P * UBUS * MS↑2 * KM↑2;
                                    2   2
(D2)                              KM  MS  P UBUS

(C3) %%T2:
        2 * P * UBUS * MS * KM  -  P * UBUS * MS↑2 * KM↑2;
                                                2   2
(D3)                    2 KM MS P UBUS - KM  MS  P UBUS

(C4) /* End of temporary variable initializations */

System%Reliability: 0;
(D4)                                  0

(C5) System%Reliability:
        2 * DBUS * KS↑2 * %%T2↑2  -  4 * DBUS * KS↑2 * %%T1 * %%T2
          +  2 * DBUS * KS↑2 * %%T1↑2  -  DBUS↑2 * KS↑4 * %%T2↑2  +  2
         * DBUS↑2 * KS↑4 * %%T1 * %%T2  -  DBUS↑2 * KS↑4 * %%T1↑2  +
        2 * %%T1  -  %%T1↑2
        ;
          2   4                          2   2         2
(D5) - DBUS  KS  (2 KM MS P UBUS - KM  MS  P UBUS)

            2                        2   2         2       2   4   4   4  2     2
 + 2 DBUS KS  (2 KM MS P UBUS - KM  MS  P UBUS)  - DBUS  KM  KS  MS  P  UBUS

            4   2   4  2     2     4   4  2     2
 + 2 DBUS KM  KS  MS  P  UBUS  - KM  MS  P  UBUS

          2   2   4   2                              2   2
 + 2 DBUS  KM  KS  MS  P UBUS (2 KM MS P UBUS - KM  MS  P UBUS)

            2   2   2                              2   2
 - 4 DBUS KM  KS  MS  P UBUS (2 KM MS P UBUS - KM  MS  P UBUS)

       2   2
 + 2 KM  MS  P UBUS

(C6) /*End of System Reliability computation*/

FACTOR(%);
        2   2                  2   2   4   2                    2   2   2
(D6) - KM  MS  P UBUS (4 DBUS  KM  KS  MS  P UBUS - 8 DBUS KM  KS  MS  P UBUS

     2   2                2       4                        2
 + KM  MS  P UBUS - 8 DBUS  KM KS  MS P UBUS + 16 DBUS KM KS  MS P UBUS

          2   4                    2
 + 4 DBUS  KS  P UBUS - 8 DBUS KS  P UBUS - 2)
```

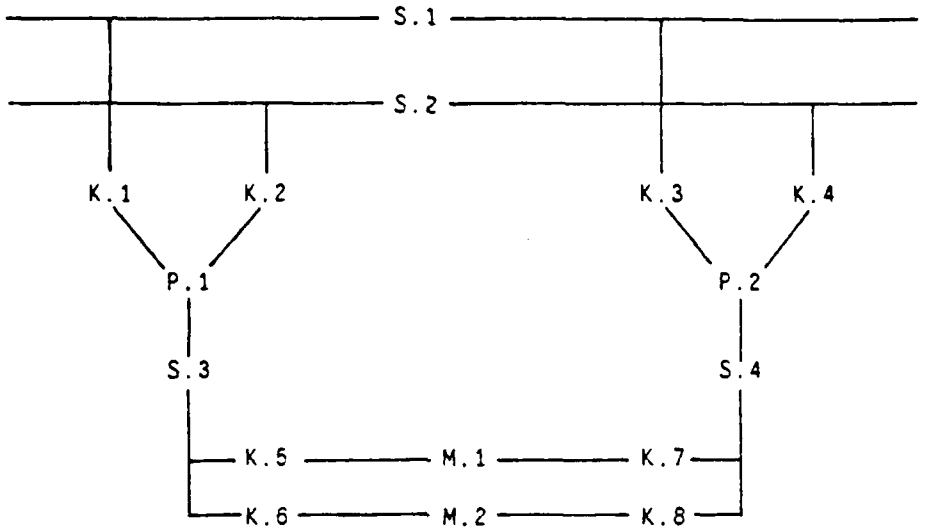
```
(D7)                                BATCH DONE
(C7) batch(kini,eghnd);

(C8) /* Reliability Function printed by LISP at 11-Jan-81 19:25:45 */

SYSREL:
+2*KM↑2*MS↑2*P*UBUS-1*KM↑4*MS↑4*P↑2*UBUS↑2+8*DBUS*KM↑2*KS↑2*MS↑2*P↑2*UBUS↑2
-16*DBUS*KM↑3*KS↑2*MS↑3*P↑2*UBUS↑2+8*DBUS*KM↑4*KS↑2*MS↑4*P↑2*UBUS↑2-4*DBUS↑2*KM
↑2*KS↑4*MS↑2*P↑2*UBUS↑2+8*DBUS↑2*KM↑3*KS↑4*MS↑3*P↑2*UBUS↑2
-4*DBUS↑2*KM↑4*KS↑4*MS↑4*P↑2*UBUS↑2;
                 2   4   4   4  2     2              4   2   4  2     2
(D8) - 4 DBUS  KM  KS  MS  P  UBUS  + 8 DBUS KM  KS  MS  P  UBUS

      4   4  2     2          2   3   4   3  2     2
 - KM  MS  P  UBUS  + 8 DBUS  KM  KS  MS  P  UBUS

              3   2   3  2     2          2   2   4   2  2     2
 - 16 DBUS KM  KS  MS  P  UBUS  - 4 DBUS  KM  KS  MS  P  UBUS

             2   2   2  2     2        2   2
 + 8 DBUS KM  KS  MS  P  UBUS  + 2 KM  MS  P UBUS

(D9)                                BATCH DONE
(C9) ratexpand(sysrel - system%reliability);
(D10)                                    0

(C10) quit();

:KILL
*
```

7.2.2 The DEC2.PMS and DEC3.PMS examples

The DEC1.PMS example shown in Figure 7-1 was a PMS structure with a Kernel composed of the component set {S.1,S.2,K.1,K.2,K.3,K.4} with the interface vertices {P.1,P.2}. P.1 and P.2 were also the root vertices of their respective Pendant Tree Subgraphs. Under the given requirements, i.e. "$\psi(1,P) \wedge \psi(2,M)$" the Kernel contained no critical components. The interface vertices were critical components but we have seen in Chapter 6 that these are counted as part of the PTS subgraphs, rather than the Kernel, for the calculation of the *intermediate result* CRPs.

The two other examples, DEC2.PMS and DEC3.PMS, in which the output of ADVISER was compared against the results of manual analysis are shown in Figures 7-3 and 7-5 along with requirements similar to those in Figure 7-1. The DEC2 example varies from the DEC1 example in that the former consists of a PMS structure which is composed entirely of a Kernel without any Pendant Tree Subgraphs. The DEC3 example on the other hand differs from the DEC1 example in that, although it has a similar structure with Kernel and PTSs, it also has critical

PMS Diagram:



Requirements:

$$\psi(1,P) \wedge \psi(1,M)$$

Figure 7-3: Example DEC2.PMS -- PMS Diagram and Requirements.

components in its Kernel. The process of checking the output of ADVISER for these examples is identical to that described above in detail for DEC1.PMS: both ADVISER and the INTERLISP program are used to obtain MACSYMA command files which are then read into the latter and the two symbolic expressions compared.

Figures 7-4 and 7-6 show the respective manually-constructed SPRBDs for the DEC2 and DEC3 examples. These may be understood in the same manner as the SPRBD for the DEC1 example. The two listings below show the MACSYMA sessions for the DEC2 and DEC3 examples in which it is demonstrated that the output of ADVISER tallies with the manual construction.

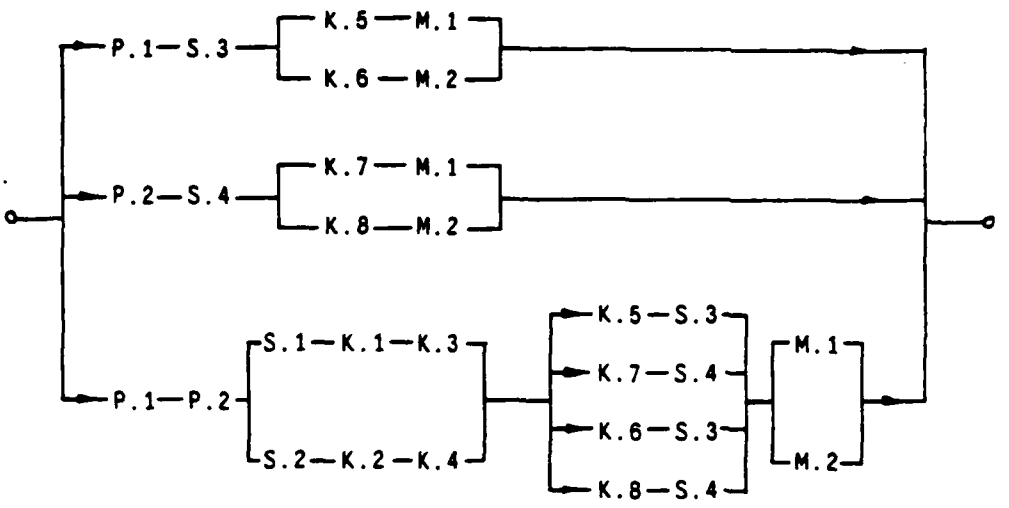


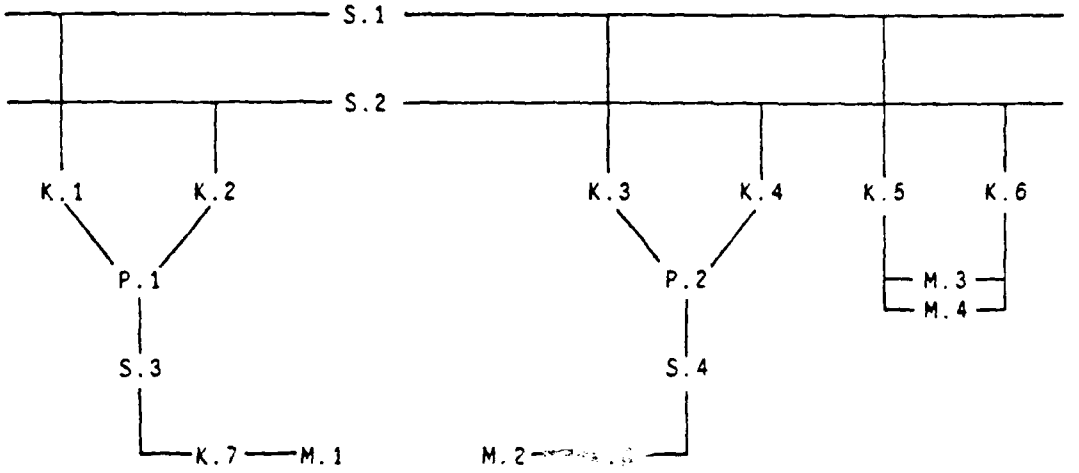
Figure 7-4: Example DEC2.PMS -- Hand-constructed SPRBD for given requirements.

Listing 6, Example DEC2.PMS

This listing shows the MACSYMA teletype session during which it is shown that the ADVISER output matches the manual construction for the DEC2 example. The ADVISER output was in the file *KINI EGADV* at the host MIT-MC on the Arpanet. The output of the INTERLISP program was in the file *KINI EGHND*. Characters typed in by the user are underlined.

```
(C8) batch(kini.egadv);

(C9) /*
-----------------------------------------------------------------------
MACSYMA Module for Reliability Function manipulation
   produced by ADVISER on Thursday,  8 Jan 81 at 15:38:44 for [4,1367]
-----------------------------------------------------------------------
Task Title: DEC2.PMS -- A Tandem/16-like architecture, Version 2

Requirements on the Structure were:

(1-OF-P AND 1-OF-MS)
-----------------------------------------------------------------------
*/

System%Reliability: 0;
(D9)                                0
```

PMS Diagram:



Requirements:

$$\psi(1,P) \wedge \psi(2,M)$$

Figure 7-5: Example DEC3.PMS -- PMS Diagram and Requirements.

```
(C10) System%Reliability:
        4 * P * UBUS * MS * KM  -  2 * P * UBUS * MS^2 * KM^2  -  2 *
        P^2 * UBUS^2 * MS * KM^2  -  2 * P^2 * UBUS^2 * MS^2 * KM^2
          +  4 * P^2 * UBUS^2 * MS^2 * KM^3  -  P^2 * UBUS^2 * MS^2 *
        KM^4
        ;
          4   2  2     2        3   2  2     2        2   2  2     2
(D10) - KM  MS  P  UBUS  + 4 KM  MS  P  UBUS  - 2 KM  MS  P  UBUS

                              2     2     2        2   2
                        - 2 KM  MS P  UBUS  - 2 KM  MS  P UBUS + 4 KM MS P UBUS

(C11) /*End of System Reliability computation*/

FACTOR(%);
                                3                    2
(D11) - KM MS P UBUS (KM  MS P UBUS - 4 KM  MS P UBUS + 2 KM MS P UBUS

                                                    + 2 KM P UBUS + 2 KM MS - 4)

(D12)                          BATCH DONE
(C12) batch(kirk.eghnd)

(C13) /* Reliability Function printed by LISP at  8-Jan-81 16:31:15 */
```

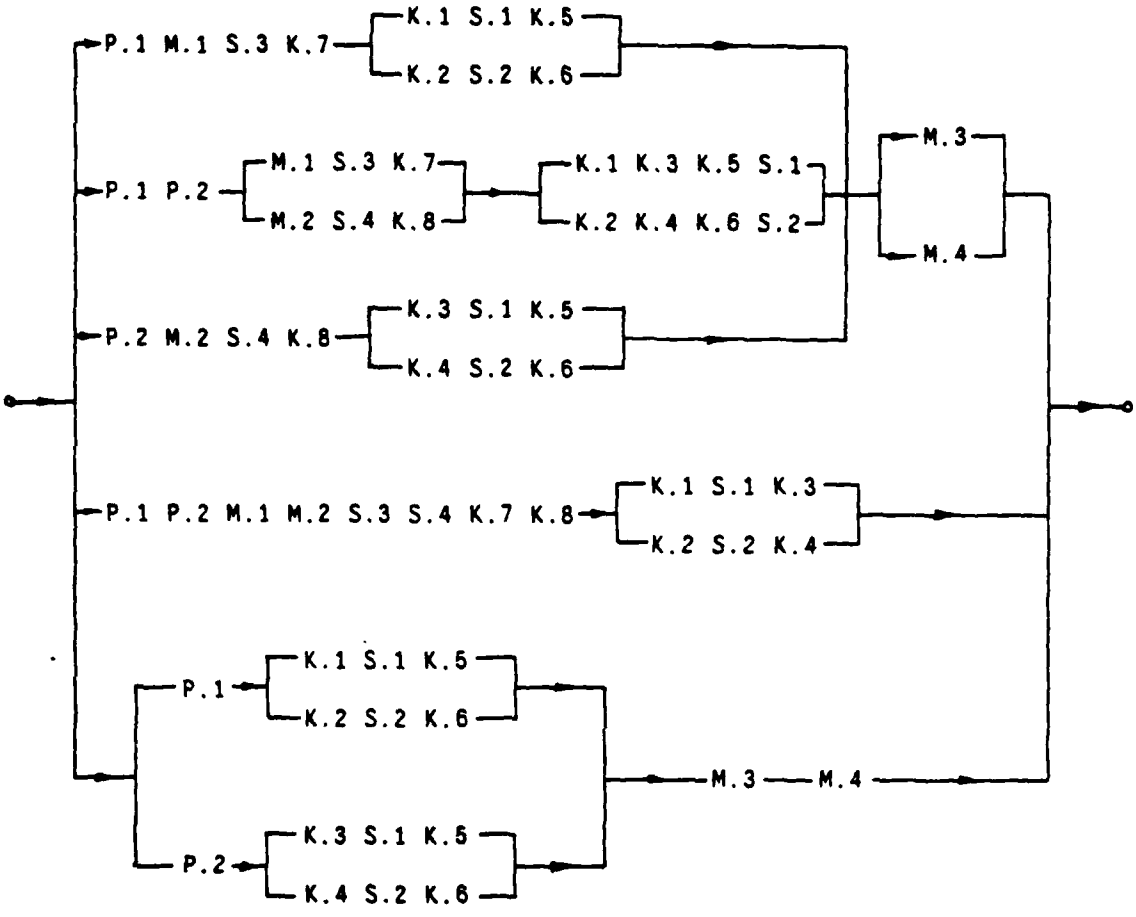


Figure 7-6: Example DEC3.PMS -- Hand-constructed SPRBD for given requirements.

```
SYSREL:
+4*KM*MS*P*UBUS-2*KM↑2*MS*P↑2*UBUS↑2-2*KM↑2*MS↑2*P*UBUS
-2*KM↑2*MS↑2*P↑2*UBUS↑2+4*KM↑3*MS↑2*P↑2*UBUS↑2-1*KM↑4*MS↑2*P↑2*UBUS↑2;

                4   2  2     2        3   2  2     2        2   2  2     2
(D13) - KM  MS  P  UBUS  + 4 KM  MS  P  UBUS  - 2 KM  MS  P  UBUS

                                 2     2     2        2   2
                       - 2 KM  MS P  UBUS  - 2 KM  MS  P UBUS + 4 KM MS P UBUS


(D14)                              BATCH DONE
(C14) system%reliability - sysrel;
(D15)                                 0

(C15) quit();
```

```
:KILL
*
```

Listing 7, Example DEC3.PMS

This listing shows the MACSYMA teletype session during which it is shown that the ADVISER output matches the manual construction. ADVISER output was in the file *KINI EGADV* and INTERLISP output was in *KINI EGHND*. Characters typed by the user are underlined.

```
(C11) batch(kini,egadv);

(C12) /*
-----------------------------------------------------------------------------
MACSYMA Module for Reliability Function manipulation
   produced by ADVISER on Thursday,  8 Jan 81 at 20:15:27 for [4,1367]
-----------------------------------------------------------------------------
Task Title: DEC3.PMS -- A Tandem/16-like architecture, Version 3

Requirements on the Structure were:

(1-OF-P AND 2-OF-MS)
-----------------------------------------------------------------------------
*/

%%T0:
        P;
(D12)                                   P

(C13) %%T1:
        P * UBUS * MS * KM;
(D13)                               KM MS P UBUS

(C14) /* End of temporary variable initializations */

System%Reliability: 0;
(D14)                                   0

(C15) System%Reliability:
        2 * DBUS * KS^2 * %%T1^2  +  8 * DBUS * KS^2 * MS * %%T1  +  4
         * DBUS * KS^2 * MS^2 * %%T0  -  8 * DBUS * KS^2 * MS^2 * %%T1
          -  8 * DBUS * KS^3 * MS * %%T1^2  -  2 * DBUS * KS^3 * MS^2
         * %%T0^2  -  DBUS^2 * KS^4 * %%T1^2  +  6 * DBUS * KS^3 *
        MS^2 * %%T1^2  -  4 * DBUS^2 * KS^4 * MS * %%T1  -  12 * DBUS^
        2 * KS^4 * MS * %%T1^2  -  2 * DBUS^2 * KS^4 * MS^2 * %%T0
          +  4 * DBUS^2 * KS^4 * MS^2 * %%T1  +  24 * DBUS^2 * KS^5 *
        MS * %%T1^2  +  8 * DBUS^2 * KS^4 * MS^2 * %%T1^2  -  2 *
        DBUS^2 * KS^4 * MS^2 * %%T0^2  -  16 * DBUS^2 * KS^5 * MS^2 *
        %%T1^2  +  4 * DBUS^2 * KS^5 * MS^2 * %%T0^2  -  8 * DBUS^2 *
        KS^6 * MS * %%T1^2  -  DBUS^2 * KS^6 * MS^2 * %%T0^2  +  5 *
        DBUS^2 * KS^6 * MS^2 * %%T1^2
        ;
```

```
                2  2  6  4  2     2          2  2  5  4  2     2
(D15) 5 DBUS  KM  KS  MS  P  UBUS  - 16 DBUS  KM  KS  MS  P  UBUS

          2  2  4  4  2     2          2  3  4  2     2
 + 8 DBUS  KM  KS  MS  P  UBUS  + 6 DBUS KM  KS  MS  P  UBUS

          2  2  6  3  2     2          2  2  6  3  2     2
 - 8 DBUS  KM  KS  MS  P  UBUS  + 24 DBUS  KM  KS  MS  P  UBUS

          2  2  4  3  2     2          2  3  3  2     2
 - 12 DBUS  KM  KS  MS  P  UBUS  - 8 DBUS KM  KS  MS  P  UBUS

        2  2  4  2  2     2          2  2  2  2     2
 - DBUS  KM  KS  MS  P  UBUS  + 2 DBUS KM  KS  MS  P  UBUS

          2     4  3                    2  3
 + 4 DBUS  KM KS  MS  P UBUS - 8 DBUS KM KS  MS  P UBUS

          2     4  2                    2  2              2  6  2  2
 - 4 DBUS  KM KS  MS  P UBUS + 8 DBUS KM KS  MS  P UBUS - DBUS  KS  MS  P

          2  5  2  2          2  4  2  2           3  2  2
 + 4 DBUS  KS  MS  P  - 2 DBUS  KS  MS  P  - 2 DBUS KS  MS  P

          2  4  2              2  2
 - 2 DBUS  KS  MS  P + 4 DBUS KS  MS  P

(C16) /*End of System Reliability computation*/

FACTOR(%);

                2  2              2  4  2       2              2  3  2       2
(D16) DBUS KS  MS  P (5 DBUS KM  KS  MS  P UBUS  - 16 DBUS KM  KS  MS  P UBUS

             2  2  2       2       2     2       2
 + 8 DBUS KM  KS  MS  P UBUS  + 6 KM  KS MS  P UBUS

             2  4          2           2  3          2
 - 8 DBUS KM  KS  MS P UBUS  + 24 DBUS KM  KS  MS P UBUS

             2  2          2       2           2          2  2       2
 - 12 DBUS KM  KS  MS P UBUS  - 8 KM  KS MS P UBUS  - DBUS KM  KS  P UBUS

       2       2              2                                  2
 + 2 KM  P UBUS  + 4 DBUS KM KS  MS UBUS - 8 KM MS UBUS - 4 DBUS KM KS  UBUS

                        4              3              2                    2
 + 8 KM UBUS - DBUS KS  P + 4 DBUS KS  P - 2 DBUS KS  P - 2 KS P - 2 DBUS KS

 + 4)

(D17)                                BATCH DONE
(C17) batch(kini.eqhnd);

(C18) /* Reliability Function printed by LISP at  8-Jan-81 17:46:30 */
```

```
SYSREL:
+4*DBUS*KS+2*MS+2*P-2*DBUS*KS+3*MS+2*P+2-2*DBUS+2*KS+4*MS+2*P
-2*DBUS+2*KS+4*MS+2*P+2-4*DBUS+2*KS+5*MS+2*P+2-1*DBUS+2*KS+6*MS+2*P+2-8*DBUS*KM
*KS+2*MS+2*P*UBUS
-8*DBUS*KM*KS+2*MS+3*P*UBUS+2*DBUS*KM+2*KS+2*MS+2*P+2*UBUS+2-8*DBUS*KM+2*KS+3*
MS+3*P+2*UBUS+2-6*DBUS*KM+2*KS+3*MS+4*P+2*UBUS+2
-4*DBUS+2*KM*KS+4*MS+2*P*UBUS+4*DBUS+2*KM*KS+4*MS+3*P*UBUS-1*DBUS+2*KM+2*KS+4*
MS+2*P+2*UBUS+2-12*DBUS+2*KM+2*KS+4*MS+3*P+2*UBUS+2
+8*DBUS+2*KM+2*KS+4*MS+4*P+2*UBUS+2+24*DBUS+2*KM+2*KS+5*MS+3*P+2*UBUS+2-16*DBUS
+2*KM+2*KS+5*MS+4*P+2*UBUS+2-8*DBUS+2*KM+2*KS+6*MS+3*P+2*UBUS+2
+5*DBUS+2*KM+2*KS+5*MS+4*P+2*UBUS+2.


                2   2   6   4  2      2            2   2   5   4  2      2
(D18) 5 DBUS  KM  KS  MS  P  UBUS  - 16 DBUS  KM  KS  MS  P  UBUS

          2   2   4   4  2      2            2   3   4  2      2
 + 8 DBUS  KM  KS  MS  P  UBUS  + 6 DBUS KM  KS  MS  P  UBUS

          2   2   6   3  2      2            2   2   5   3  2      2
 - 8 DBUS  KM  KS  MS  P  UBUS  + 24 DBUS  KM  KS  MS  P  UBUS

          2   2   4   3  2      2            2   3   3  2      2
 - 12 DBUS  KM  KS  MS  P  UBUS  - 8 DBUS KM  KS  MS  P  UBUS

        2   2   4   2  2      2            2   2   2  2      2
 - DBUS  KM  KS  MS  P  UBUS  + 2 DBUS KM  KS  MS  P  UBUS

          2       4   3                        2   3
 + 4 DBUS  KM KS  MS  P UBUS - 8 DBUS KM KS  MS  P UBUS

          2       4   2                        2   2                2   6   2  2
 - 4 DBUS  KM KS  MS  P UBUS + 8 DBUS KM KS  MS  P UBUS - DBUS  KS  MS  P

          2   5   2  2          2   4   2  2            3   2  2
 + 4 DBUS  KS  MS  P  - 2 DBUS  KS  MS  P  - 2 DBUS KS  MS  P

          2   4   2              2   2
 - 2 DBUS  KS  MS  P + 4 DBUS KS  MS  P

(D19)                              BATCH DONE

(C20) ratexpand(sysrel - system%reliability);
(D20)                                  0

(C21) quit();

:KILL
•
```

7.3 Comparison to published results

The work described in [Siewiorek 78] provided a useful opportunity to test the output of ADVISER for correctness. For convenience we shall henceforth refer to [Siewiorek 78] as SENET which is the acronym for the subject of that report. SENET presented general reliability functions derived by hand for each of a set of five multiprocessor architectures. In the SENET work a Fortran program was constructed for each architecture to compute numerical values of its reliability given architectural parameters and operational requirements. Assuming a very small chance of the exact same computational error occurring in both the SENET work and the output of ADVISER, a good test of the latter is to numerically evaluate its output and compare it to the output from SENET programs. Although disagreement in the two sets of results proves at best that one of the outputs is in error, agreement on the other hand engenders substantial confidence in both. This is especially true since the methods of arriving at the system reliability functions in the two cases are so different.

Following sections describe the results of the comparison. The results of only one or two tests per architecture are actually included here but several tests were satisfactorily carried out with each architecture. In those cases where the outputs of the two programs did not match, the known or suspected reasons are provided. Mismatches in the case of the Global Bus architecture (Section 7.3.4) were traced to errors in the SENET program for the architecture and the ADVISER output was shown to be correct by comparing to manual computation. Mismatches in the case of the Pluribus architecture (Section 7.3.5) and two test cases of the Cm* architecture (Section 7.3.1) were caused by known deficiencies in the ADVISER algorithms. *It is to be emphasized here that these are not serious deficiences; they result from incomplete implementation in ADVISER of the ideas discussed in Chapters 4 and 6 and manifest themselves in infrequent cases.* A mismatch in two cases of the Tandem architecture (Section 7.3.3) are strongly suspected to arise from errors in the SENET program for the architecture since manual computation once more proved ADVISER to be correct. All other tests resulted in successful matches.

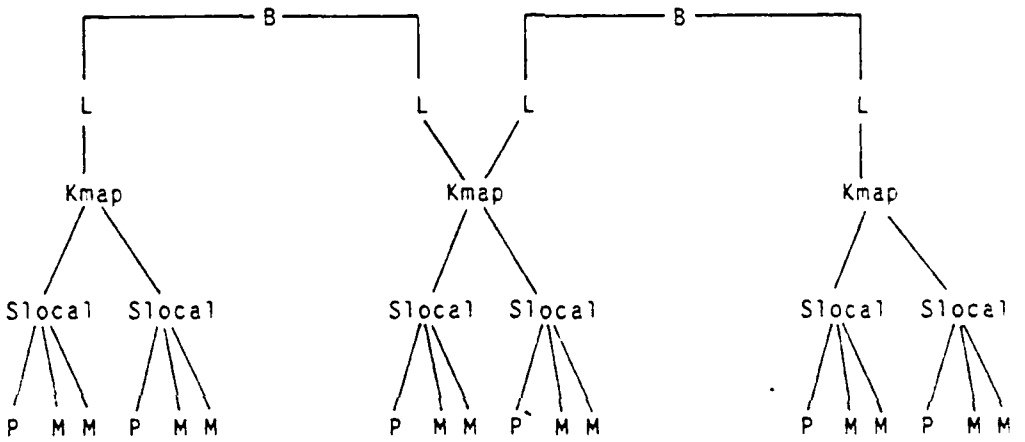
NOTE:

- *In the rest of this chapter, the phrase "successful match" or "successful test" will be used to mean that the absolute differences between the numerical values of the output of the two programs were within one percent of the SENET values.*

- *Each of the SENET programs was constructed to compute the reliability of the architecture it addresses for a range of values along the time axis, starting at t = 200 hrs. and 200 hrs. apart. Computation continues until 200 values have been computed or the reliability has fallen below 10^{-4}. In displaying the results of*

the comparison of ADVISER and SENET values in succeeding pages, only representative samples will be shown if the total number of values is too large.

- *In all of the comparisons, as in SENET, all components were presumed to have exponential distributions for their times to failure. The failure rates used as inputs to ADVISER were taken directly from SENET. Note, however, that the ADVISER output can easily be modified for other failure distributions.*



Key

B	Intercluster Bus	Slocal	Local switch
L	Intercluster Bus Interface	P	Processor
Kmap	Mapping Controller	M	Memory

Figure 7-7: Cm* architecture used for ADVISER test

7.3.1 The Cm* architecture

The Cm* multiprocessor architecture is based on the LSI-11 microcomputer and is described in [Swan 77]. Figure 7-7 shows the version of the architecture which was used for the ADVISER test. It consists of three clusters of two computer modules (Cm's) each. Each Cm is composed of one processor with two memories. The memories in the structure collectively realize the virtual address space shared by the processors. The KMAPs in Figure

7-7 are mapping controllers which allow processors in Cms to access memory elsewhere in the cluster or in other clusters via the Intercluster Buses (B in the figure). The components marked L in the figure are the interfaces from the KMAPs to Intercluster Buses.

In one test using the Cm* architecture, from a total of six processors and 12 memories the test imposed the requirement that five processors and ten memories be functional. For these configuration parameters the SENET Cm* program was run to obtain numerical system reliability values. ADVISER was then run on the same problem with the requirement expression "$\psi(5,P) \wedge \psi(10,M)$". The Fortran output of ADVISER for the problem was compiled and called repeatedly by a Fortran driver program to obtain reliability values for the same points along the time axis as in the corresponding SENET case. Figure 7-8 shows the results of the test, for samples of the time values, in five columns. The fifth and last column lists the differences in the fourth column as a percentage of corresponding SENET values.

In this fashion all combinations of requirements up to a maximum of 6 processors and 12 memories were tried. The results matched SENET results except in two cases viz. the requirements "$\psi(1,P) \wedge \psi(1,M)$" and "$\psi(1,P) \wedge \psi(2,M)$". See Figure 7-9 which shows the mismatch for the latter of these two requirements. The reason for this mismatch is a known deficiency in the TREEREL algorithm of ADVISER. When the above two requirements are imposed on the given Cm* structure there are functional states of the system in which the requirements are satisfied by a single Cm which has one processor and two memories within it. In this case, theoretically, no other components outside the Cm are necessary to cause the system state to be functional. However, the TREEREL algorithm was used to generate partial result CRPs before system CRP was computed. As described in Chapter 5 the TREEREL algorithm assumes that all components on the path from any currently active critical component in the PTS to the root of the PTS, upto and including the root vertex, need to be functional in order to satisfy the Communication Axiom. If the requirements are entirely satisfied by some subtree of a PTS, as in the case of the two simple requirements above, then there is no need for components up to the root vertex of the PTS to be functional. In terms of the Cm* example, since the requirements are met by a single Cm there is no need for the Kmap to be functional. However, the TREEREL algorithm does take into account the Kmap reliability even in this case and thus causes the reliability to differ from SENET results. This situation was anticipated in Section 5.3.

Note, however, that this problem may currently be sidestepped in an *ad hoc* fashion by forcing all the Kmaps to be part of the Kernel by assigning them trivial Internal Port Connection Matrices (see Section 6.9.1.3 on Page 167; for this example all the links, L in

Time	SENET	ADVISER	(SENET-ADVISER)	% Diff
200.0	0.9030371E+00	0.9030333E+00	0.3822148E-05	0.00
400.0	0.8067455E+00	0.8067406E+00	0.5930662E-05	0.00
600.0	0.7131108E+00	0.7131026E+00	0.8165836E-05	0.00
800.0	0.6240257E+00	0.6240165E+00	0.9246171E-05	0.00
1000.0	0.5409821E+00	0.5409721E+00	0.1004338E-04	0.00
1200.0	0.4649715E+00	0.4649610E+00	0.1053885E-04	0.00
1400.0	0.3965082E+00	0.3964977E+00	0.1049787E-04	0.00
1600.0	0.3357063E+00	0.3356961E+00	0.1024827E-04	0.00
1800.0	0.2823718E+00	0.2823621E+00	0.9689480E-05	0.00
2000.0	0.2360937E+00	0.2360848E+00	0.8916482E-05	0.00
2200.0	0.1963226E+00	0.1963144E+00	0.8180737E-05	0.00
2400.0	0.1624339E+00	0.1624266E+00	0.7305294E-05	0.00
2600.0	0.1337768E+00	0.1337702E+00	0.6550923E-05	0.00
2800.0	0.1097086E+00	0.1097028E+00	0.5849637E-05	0.01
3000.0	0.8961848E-01	0.8961337E-01	0.5110167E-05	0.01
3200.0	0.7294213E-01	0.7293771E-01	0.4420988E-05	0.01
3400.0	0.5916942E-01	0.5916560E-01	0.3816094E-05	0.01
3600.0	0.4784723E-01	0.4784397E-01	0.3260560E-05	0.01
3800.0	0.3857868E-01	0.3857610E-01	0.2782325E-05	0.01
4000.0	0.3102126E-01	0.3101890E-01	0.2355548E-05	0.01
4200.0	0.2488069E-01	0.2487870E-01	0.1987210E-05	0.01
4400.0	0.1990800E-01	0.1990634E-01	0.1662644E-05	0.01
4600.0	0.1589346E-01	0.1589207E-01	0.1392560E-05	0.01
4800.0	0.1266170E-01	0.1266055E-01	0.1154258E-05	0.01
5000.0	0.1006703E-01	0.1006608E-01	0.9514624E-06	0.01
5200.0	0.7989052E-02	0.7988264E-02	0.7878989E-06	0.01
5400.0	0.6328749E-02	0.6328100E-02	0.6491900E-06	0.01
5600.0	0.5005073E-02	0.5004541E-02	0.5317270E-06	0.01
5800.0	0.3951945E-02	0.3951509E-02	0.4362082E-06	0.01
6000.0	0.3115689E-02	0.3115335E-02	0.3544647E-06	0.01
6200.0	0.2452862E-02	0.2452573E-02	0.2891466E-06	0.01
6400.0	0.1928403E-02	0.1928168E-02	0.2346205E-06	0.01
6600.0	0.1514107E-02	0.1513917E-02	0.1903391E-06	0.01
6800.0	0.1187340E-02	0.1187187E-02	0.1534354E-06	0.01
7000.0	0.9299902E-03	0.9298667E-03	0.1235312E-06	0.01
7200.0	0.7275941E-03	0.7274947E-03	0.9943324E-07	0.01
7400.0	0.5686287E-03	0.5685488E-03	0.7991912E-07	0.01
7600.0	0.4439328E-03	0.4438688E-03	0.6397022E-07	0.01
7800.0	0.3462370E-03	0.3461858E-03	0.5120091E-07	0.01
8000.0	0.2697834E-03	0.2697425E-03	0.4093090E-07	0.02
8200.0	0.2100194E-03	0.2099866E-03	0.3280002E-07	0.02
8400.0	0.1633507E-03	0.1633246E-03	0.2610068E-07	0.02
8600.0	0.1269448E-03	0.1269241E-03	0.2068737E-07	0.02

Figure 7-8: Comparison of ADVISER and SENET results for Figure 7-7, Cm*, 5 P, 10 M required.

Time	SENET	ADVISER	(SENET-ADVISER)	% Diff.
200.0	0.9999730E+00	0.9999722E+00	0.8344650E-06	0.00
400.0	0.9997945E+00	0.9997808E+00	0.1372397E-04	0.00
600.0	0.9993369E+00	0.9992698E+00	0.6711483E-04	0.01
800.0	0.9984970E+00	0.9982957E+00	0.2013370E-03	0.02
1000.0	0.9971916E+00	0.9967242E+00	0.4673526E-03	0.05
1200.0	0.9953556E+00	0.9944353E+00	0.9202883E-03	0.09
1400.0	0.9929393E+00	0.9913239E+00	0.1616353E-02	0.16
1600.0	0.9899059E+00	0.9872989E+00	0.2606966E-02	0.26
1800.0	0.9862300E+00	0.9822857E+00	0.3944293E-02	0.40
2000.0	0.9818953E+00	0.9762253E+00	0.5669951E-02	0.58
2200.0	0.9768935E+00	0.9690743E+00	0.7819161E-02	0.80
2400.0	0.9712227E+00	0.9608054E+00	0.1041731E-01	1.07
2600.0	0.9648860E+00	0.9514064E+00	0.1347961E-01	1.40
2800.0	0.9578912E+00	0.9408782E+00	0.1701299E-01	1.78
3000.0	0.9502490E+00	0.9292360E+00	0.2101298E-01	2.21
3200.0	0.9419730E+00	0.9165066E+00	0.2546640E-01	2.70
3400.0	0.9330787E+00	0.9027279E+00	0.3035083E-01	3.25
3600.0	0.9235832E+00	0.8879471E+00	0.3563607E-01	3.86
3800.0	0.9135049E+00	0.8722199E+00	0.4128495E-01	4.52
4000.0	0.9028628E+00	0.8556089E+00	0.4725389E-01	5.23
4200.0	0.8916769E+00	0.8381827E+00	0.5349422E-01	6.00
4400.0	0.8799678E+00	0.8200143E+00	0.5995355E-01	6.81
4600.0	0.8677563E+00	0.8011793E+00	0.6657703E-01	7.67
4800.0	0.8550640E+00	0.7817565E+00	0.7330754E-01	8.57
5000.0	0.8419129E+00	0.7618252E+00	0.8008774E-01	9.51
5200.0	0.8283256E+00	0.7414650E+00	0.8686056E-01	10.49
5400.0	0.8143252E+00	0.7207553E+00	0.9356992E-01	11.49
5600.0	0.7999355E+00	0.6997733E+00	0.1001622E+00	12.52
5800.0	0.7851810E+00	0.6785947E+00	0.1065863E+00	13.57
6000.0	0.7700870E+00	0.6572924E+00	0.1127946E+00	14.65
14800.0	0.1525339E+00	0.7468458E-01	0.7784932E-01	51.04
15000.0	0.1448945E+00	0.7032622E-01	0.7456828E-01	51.46
15200.0	0.1375726E+00	0.6620502E-01	0.7136758E-01	51.88
15400.0	0.1305604E+00	0.6230972E-01	0.6825068E-01	52.28
15600.0	0.1238498E+00	0.5862943E-01	0.6522037E-01	52.66
15800.0	0.1174324E+00	0.5515366E-01	0.6227874E-01	53.03
16000.0	0.1112999E+00	0.5187227E-01	0.5942763E-01	53.39
16200.0	0.1054434E+00	0.4877551E-01	0.5666789E-01	53.74
16400.0	0.9985439E-01	0.4585401E-01	0.5400038E-01	54.08
16600.0	0.9452401E-01	0.4309878E-01	0.5142523E-01	54.40
16800.0	0.8944351E-01	0.4050120E-01	0.4894231E-01	54.72
17000.0	0.8460410E-01	0.3805299E-01	0.4655111E-01	55.02
17200.0	0.7999710E-01	0.3574626E-01	0.4425084E-01	55.32
17400.0	0.7561383E-01	0.3357347E-01	0.4204036E-01	55.60
17600.0	0.7144576E-01	0.3152738E-01	0.3991838E-01	55.87
17800.0	0.6748447E-01	0.2960114E-01	0.3788333E-01	56.14
18000.0	0.6372169E-01	0.2778819E-01	0.3593350E-01	56.39
18200.0	0.6014929E-01	0.2608228E-01	0.3406701E-01	56.64
18400.0	0.5675932E-01	0.2447747E-01	0.3228185E-01	56.87
18600.0	0.5354400E-01	0.2296812E-01	0.3057588E-01	57.10
18800.0	0.5049577E-01	0.2154886E-01	0.2894691E-01	57.33
19000.0	0.4760725E-01	0.2021460E-01	0.2739265E-01	57.54
19200.0	0.4487128E-01	0.1896050E-01	0.2591078E-01	57.74
19400.0	0.4228092E-01	0.1778198E-01	0.2449894E-01	57.94
19600.0	0.3982942E-01	0.1667470E-01	0.2315472E-01	58.13
19800.0	0.3751029E-01	0.1563464E-01	0.2187575E-01	58.32
20000.0	0.3531723E-01	0.1465761E-01	0.2065962E-01	58.50

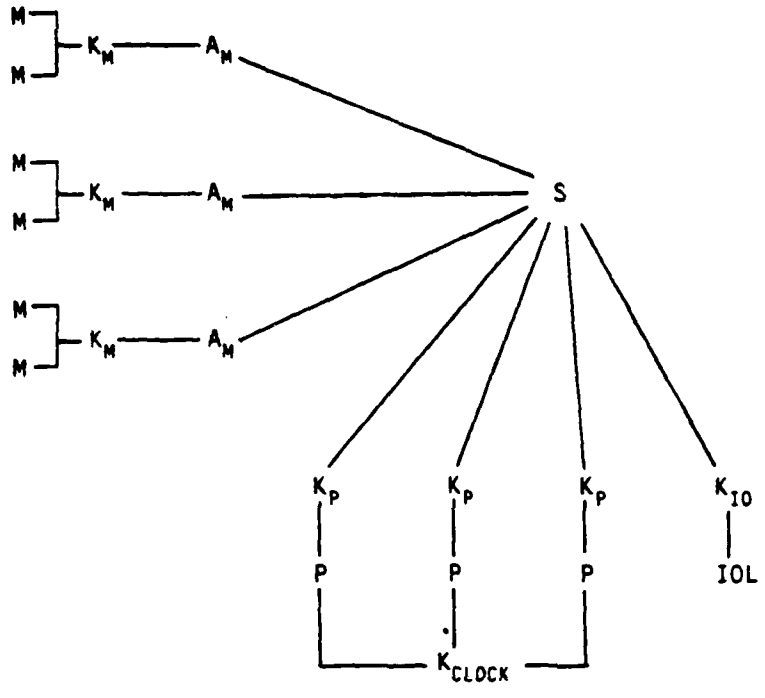
Figure 7-9: Comparison of ADVISER and SENET results for Figure 7-7, Cm*, 1 P, 2 M required.

Figure 7-7, were forced into the Kernel). This is not always recommended with the current version of ADVISER, however, since this usually results in a larger Kernel, the number of PTSs in the graph is higher and thus the number of cases of feasible compositions rises thereby causing a much higher computation time for the problem. In fact, although with high likelihood the ADVISER test would have succeeded if all Kmaps were included in the Kernel, doing so generated so many cases to be analyzed that the current ADVISER version would have used extreme amounts of CPU time for the solution.

7.3.2 The C.mmp architecture

The next of the SENET examples chosen for comparison with ADVISER was the C.mmp architecture [Wulf 72]. The architecture consists of an N x M crosspoint switch which has N memory ports and M processor ports. Hence each processor is able to access each memory port. In the original C.mmp architecture contention for access to the same memory port was resolved by queueing requests at the port. Figure 7-10 shows the model treated by SENET. On the processor side of the crosspoint switch the switch ports may have I/O lines attached to them through direct-memory-access I/O controllers. SENET also treats two cases of switch reliability separately. In one case the individual crosspoint reliabilities are lumped and the switch is considered an indivisible component. In the other case the individual crosspoint reliabilities and the actual distributed structure of the switch are taken into account. For further details the reader is referred to [Siewiorek 78]. The particular example chosen for testing here is shown in Figure 7-10 and consists of six memories, two on each of three memory ports, and three processors and one I/O controller each on their own ports on the processor side of the switch. The K.clock is a system wide clock used for processor synchronization and its functioning is essential to the system. An IPCM (see Chapter 6) was assigned to the crosspoint switch in the lumped case which allowed communication only from processors to memories and *vice versa*.

Figure 7-11 shows parts of the comparison between ADVISER and SENET results for the architecture of Figure 7-10 with the switch treated as lumped and the requirement $\psi(2,P) \wedge \psi(2,M) \wedge \psi(1,K.io) \wedge \psi(1,K.clock)$. Similarly, Figure 7-12 shows parts of the comparison between ADVISER and SENET results for the architecture of Figure 7-10 with the switch treated as distributed and the same requirement as for the case of the lumped switch. The small differences of less than one percent in this case possibly arise from a slightly different way of assigning the crosspoint failure rates to the switch multiplexers in the SENET program versus ADVISER. The implementation of the switch actually realizes the conceptual NxM



Key

A_M	Memory Arbiter	K_M	Memory Control
K_P	"Relocation Box"	K_{CLOCK}	System Clock
IOL	I/O Line	P	Processor
S	Crosspoint Switch	M	Memory

Figure 7-10: C.mmp architecture for ADVISER test.

crosspoints as a set of M N-to-1 multiplexers in the memory-to-processor paths and N M-to-1 multiplexers in the processor-to-memory paths (see [Siewiorek 78]).

7.3.3 The Tandem architecture

The third example treated by SENET is the Tandem-16 NonStop[37] architecture which is described in detail in [Katzman 77]. Figure 7-13(a) shows the version of the Tandem

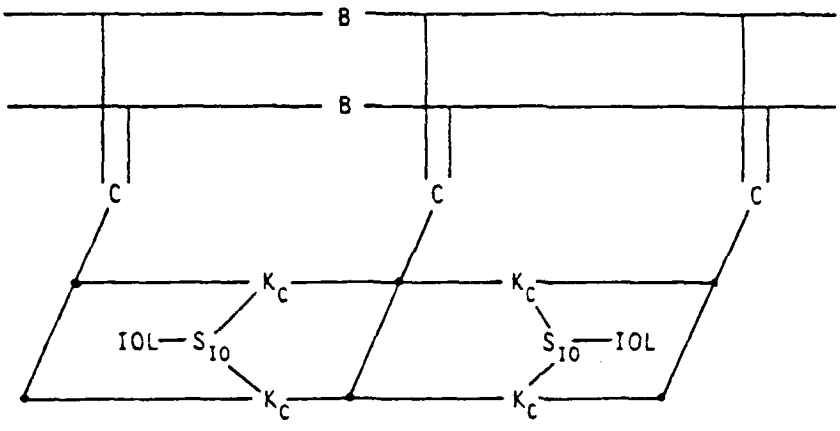
[37] Tandem-16 and NonStop are registered trademarks of Tandem Computers Inc.

Time	SENET	ADVISER	(SENET-ADVISER)	% Diff.
200.0	0.9911113E+00	0.9911112E+00	0.8195639E-07	0.00
400.0	0.9822786E+00	0.9822788E+00	-0.1639128E-06	0.00
600.0	0.9735020E+00	0.9735019E+00	0.8195639E-07	0.00
800.0	0.9647815E+00	0.9647815E+00	-0.2235174E-07	0.00
1000.0	0.9561173E+00	0.9561159E+00	0.3874302E-06	0.00
1200.0	0.9475094E+00	0.9475095E+00	-0.1266599E-06	0.00
1400.0	0.9389579E+00	0.9389580E+00	-0.1490116E-06	0.00
1600.0	0.9304629E+00	0.9304627E+00	0.2384186E-06	0.00
1800.0	0.9220243E+00	0.9220242E+00	0.1117587E-06	0.00
2000.0	0.9136422E+00	0.9136419E+00	0.2682209E-06	0.00
2200.0	0.9053156E+00	0.9053158E+00	-0.7823110E-06	0.00
2400.0	0.8970475E+00	0.8970462E+00	0.1288950E-05	0.00
2600.0	0.8888348E+00	0.8888339E+00	0.8568156E-06	0.00
2800.0	0.8806785E+00	0.8806773E+00	0.1214446E-05	0.00
10200.0	0.6173163E+00	0.6173077E+00	0.8590519E-05	0.00
10400.0	0.6111867E+00	0.6111779E+00	0.8776784E-05	0.00
10600.0	0.6051059E+00	0.6050972E+00	0.8709729E-05	0.00
10800.0	0.5990737E+00	0.5990646E+00	0.9097159E-05	0.00
11000.0	0.5930897E+00	0.5930806E+00	0.9074507E-05	0.00
11200.0	0.5871538E+00	0.5871444E+00	0.9350479E-05	0.00
11400.0	0.5812656E+00	0.5812561E+00	0.9506941E-05	0.00
11600.0	0.5754249E+00	0.5754152E+00	0.9693205E-05	0.00
11800.0	0.5696314E+00	0.5696217E+00	0.9700656E-05	0.00
12000.0	0.5638848E+00	0.5638751E+00	0.9737909E-05	0.00
12200.0	0.5581849E+00	0.5581748E+00	0.1009564E-04	0.00
12400.0	0.5525313E+00	0.5525212E+00	0.1008809E-04	0.00
12600.0	0.5469239E+00	0.5469136E+00	0.1033396E-04	0.00
12800.0	0.5413623E+00	0.5413518E+00	0.1050532E-04	0.00
27500.0	0.2390987E+00	0.2390896E+00	0.9100884E-05	0.00
27800.0	0.2362678E+00	0.2362587E+00	0.9058043E-05	0.00
28000.0	0.2334649E+00	0.2334559E+00	0.9026378E-05	0.00
28200.0	0.2306897E+00	0.2306808E+00	0.8936834E-05	0.00
28400.0	0.2279420E+00	0.2279332E+00	0.8784235E-05	0.00
28600.0	0.2252217E+00	0.2252130E+00	0.8674338E-05	0.00
28800.0	0.2225286E+00	0.2225200E+00	0.8590519E-05	0.00
29000.0	0.2198625E+00	0.2198539E+00	0.8575618E-05	0.00
29200.0	0.2172231E+00	0.2172146E+00	0.8450821E-05	0.00
29400.0	0.2146103E+00	0.2146019E+00	0.8363277E-05	0.00
29600.0	0.2120238E+00	0.2120156E+00	0.8186325E-05	0.00
29800.0	0.2094636E+00	0.2094554E+00	0.8180737E-05	0.00
30000.0	0.2069293E+00	0.2069213E+00	0.8026138E-05	0.00
37800.0	0.1263744E+00	0.1263700E+00	0.4375353E-05	0.00
38000.0	0.1247286E+00	0.1247243E+00	0.4314817E-05	0.00
38200.0	0.1231014E+00	0.1230972E+00	0.4225411E-05	0.00
38400.0	0.1214927E+00	0.1214885E+00	0.4170462E-05	0.00
38600.0	0.1199023E+00	0.1198982E+00	0.4119240E-05	0.00
38800.0	0.1183300E+00	0.1183260E+00	0.4023314E-05	0.00
39000.0	0.1167757E+00	0.1167718E+00	0.3945082E-05	0.00
39200.0	0.1152392E+00	0.1152353E+00	0.3855575E-05	0.00
39400.0	0.1137204E+00	0.1137166E+00	0.3815629E-05	0.00
39600.0	0.1122190E+00	0.1122153E+00	0.3672205E-05	0.00
39800.0	0.1107350E+00	0.1107314E+00	0.3603287E-05	0.00
40000.0	0.1092682E+00	0.1092647E+00	0.3537163E-05	0.00

Figure 7-11: Comparison of ADVISER and SENET results for Figure 7-10 C.mmp, lumped switch, 2 P, 2 M and 1 K.io required.

Time	SENET	ADVISER	(SENET-ADVISER)	% Diff.
200.0	0.9975381E+00	0.9976528E+00	-0.1147017E-03	0.01
400.0	0.9950595E+00	0.9952881E+00	-0.2286360E-03	0.02
600.0	0.9925644E+00	0.9929053E+00	-0.3409162E-03	0.03
800.0	0.9900534E+00	0.9905054E+00	-0.4519969E-03	0.05
1000.0	0.9875266E+00	0.9880890E+00	-0.5624220E-03	0.06
1200.0	0.9849844E+00	0.9856559E+00	-0.6716357E-03	0.07
1400.0	0.9824272E+00	0.9832067E+00	-0.7794574E-03	0.08
1600.0	0.9798553E+00	0.9807418E+00	-0.8865297E-03	0.09
1800.0	0.9772690E+00	0.9782609E+00	-0.9919032E-03	0.10
2000.0	0.9746688E+00	0.9757654E+00	-0.1096606E-02	0.11
2200.0	0.9720548E+00	0.9732548E+00	-0.1199968E-02	0.12
2400.0	0.9694274E+00	0.9707297E+00	-0.1302280E-02	0.13
2600.0	0.9667867E+00	0.9681900E+00	-0.1403287E-02	0.15
9200.0	0.8737929E+00	0.8778365E+00	-0.4043624E-02	0.46
9400.0	0.8708335E+00	0.8749353E+00	-0.4101843E-02	0.47
9600.0	0.8678675E+00	0.8720262E+00	-0.4158720E-02	0.48
9800.0	0.8648948E+00	0.8691091E+00	-0.4214294E-02	0.49
10000.0	0.8619156E+00	0.8661845E+00	-0.4268914E-02	0.50
10200.0	0.8589303E+00	0.8632521E+00	-0.4321851E-02	0.50
10400.0	0.8559386E+00	0.8603122E+00	-0.4373640E-02	0.51
15600.0	0.7762822E+00	0.7815654E+00	-0.5283222E-02	0.68
15800.0	0.7731580E+00	0.7784604E+00	-0.5302370E-02	0 69
16000.0	0.7700303E+00	0.7753506E+00	-0.5320296E-02	0.69
16200.0	0.7668991E+00	0.7722361E+00	-0.5336985E-02	0.70
16400.0	0.7637644E+00	0.7691172E+00	-0.5352832E-02	0.70
16600.0	0.7606265E+00	0.7659939E+00	-0.5367398E-02	0.71
21400.0	0.6845764E+00	0.6900008E+00	-0.5424440E-02	0.79
21600.0	0.6813883E+00	0.6868041E+00	-0.5415820E-02	0.79
21800.0	0.6781997E+00	0.6836061E+00	-0.5406417E-02	0.80
22000.0	0.6750106E+00	0.6804069E+00	-0.5396277E-02	0.80
22200.0	0.6718211E+00	0.6772065E+00	-0.5385429E-02	0.80
22400.0	0.6686315E+00	0.6740052E+00	-0.5373731E-02	0.80
22600.0	0.6654417E+00	0.6708030E+00	-0.5361326E-02	0.81
25200.0	0.6240197E+00	0.6291604E+00	-0.5140752E-02	0.82
25400.0	0.6208412E+00	0.6259608E+00	-0.5119599E-02	0.82
25600.0	0.6176643E+00	0.6227623E+00	-0.5098045E-02	0.83
25800.0	0.6144894E+00	0.6195652E+00	-0.5075775E-02	0.83
26000.0	0.6113165E+00	0.6163695E+00	-0.5053014E-02	0.83
28800.0	0.5671748E+00	0.5718629E+00	-0.4688144E-02	0 83
29000.0	0.5640468E+00	0.5687060E+00	-0.4659235E-02	0 83
29200.0	0.5609228E+00	0.5655528E+00	-0.4630029E-02	0.83
29400.0	0.5578030E+00	0.5624034E+00	-0.4600435E-02	0.82
29600.0	0.5546875E+00	0.5592580E+00	-0.4570462E-02	0.82
32200.0	0.5146284E+00	0.5187874E+00	-0.4158989E-02	0.81
32400.0	0.5115852E+00	0.5157112E+00	-0.4125968E-02	0.81
32600.0	0.5085482E+00	0.5126408E+00	-0.4092634E-02	0.80
32800.0	0.5055172E+00	0.5095765E+00	-0.4059300E-02	0.80
33000.0	0.5024926E+00	0.5065184E+00	-0.4025780E-02	0.80
33200.0	0.4994744E+00	0.5034665E+00	-0.3992125E-02	0.80
33400.0	0.4964627E+00	0.5004211E+00	-0.3958356E-02	0.80
33600.0	0.4934577E+00	0.4973821E+00	-0.3924429E-02	0.80
39000.0	0.4152083E+00	0.4182035E+00	-0.2995219E-02	0.72
39200.0	0.4124301E+00	0.4153913E+00	-0.2961207E-02	0.72
39400.0	0.4096613E+00	0.4125886E+00	-0.2927266E-02	0.71
39600.0	0.4069020E+00	0.4097954E+00	-0.2893418E-02	0.71
39800.0	0.4041523E+00	0.4070119E+00	-0.2859626E-02	0.71
40000.0	0.4014123E+00	0.4042382E+00	-0.2825856E-02	0.70

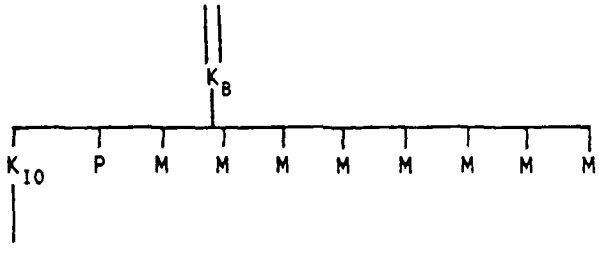
Figure 7-12: Comparison of ADVISER and SENET results for Figure 7-10. C.mmp, distributed switch, 2 P, 2 M and 1 K.io required.



Key

B	Dynabus	IOL	I/O Line
C	Computer	S_{IO}	I/O Switch
K_C	Communications Control		

(a)



Key

K_{IO}	I/O Control	P	Processor
K_B	Dynabus Control	M	Memory

(b)

Figure 7-13: Tandem-16 architecture for ADVISER test.
(a) PMS diagram (b) Detail of Computer.

architecture used for the test. There are three computers which communicate via a duplicated fast bus termed the Dynabus. Each computer is composed of a processor, local memory, a Dynabus control and an I/O channel (Figure 7-13(b)). The computers communicate with I/O lines via dual-ported I/O switches which are connected to dual-ported (possibly multi-ported) communications controllers. This arrangement gives the architecture high availability.

The Tandem example provided an opportunity for using the facility in ADVISER where independent symmetric sub-structures of the original PMS structure are submitted as separate problems to ADVISER and the solutions are used in conjunction. In the Tandem example, the computers were taken to be identical and each computer was taken by SENET to have eight local memories of which at least six were required to function for a reliable computer. In order for the architecture to be functional the requirement was that at least two of the three computers and both the I/O lines should be functional. The Tandem example was split into two problems for ADVISER. The first problem is to compute the reliability of a single computer with the requirement that six of the eight available local memories, the processor, the Dynabus control and the I/O channel should function. The second problem considers the Tandem architecture with the computers as indivisible entities and computes the reliability assuming that two of the computers and both the I/O lines need to function. The reliability of components of type "computer" in the second problem is taken to be what was obtained as a solution of first problem. ADVISER provides the capability of asserting that the reliability of a component type will be provided by an external Fortran (or Sail) function. Then, at the appropriate point in the Fortran (or Sail) output for the example, calls are provided to the external function in order to obtain the reliability of the substructure, in this case a computer.

Figure 7-14 shows the results of the comparison for the Tandem architecture of Figure 7-13 for the requirement "ψ(2,C) $\wedge$ ψ(2,IOL)". For each computer the requirement was "ψ(6,M) $\wedge$ ψ(1,P) $\wedge$ ψ(1,IOC) $\wedge$ ψ(1,Kb)". The tests were also successful for the case of a four-computer Tandem model which contained four computers and three I/O lines connected in regular fashion in extension of Figure 7-13(a).

There were two cases in which the output of ADVISER did not match that of SENET for the example of Figure 7-13. The requirements in these cases were respectively "ψ(1,C) $\wedge$ ψ(1,IOL)" and "ψ(2,C) $\wedge$ ψ(1,IOL)". Figure 7-15 shows the results of the comparison in the first for the first of these two requirements. In both these, however, cases hand computation using the LISP-MACSYMA combination described in Section 7.2 showed that the ADVISER output was correct. Since SENET is not too clear on this point, the discrepancies are thought

Time	SENET	ADVISER	(SENET-ADVISER	% Diff
200.0	0.9872584E+00	0.9872582E+00	0.1564622E-06	0.00
400.0	0.9731240E+00	0.9731238E+00	0.1937151E-06	0.00
600.0	0.9577239E+00	0.9577239E+00	-0.4470348E-07	0.00
800.0	0.9411763E+00	0.9411760E+00	0.3501773E-06	0.00
1000.0	0.9235914E+00	0.9235910E+00	0.3501773E-06	0.00
1200.0	0.9050732E+00	0.9050734E+00	-0.1713634E-06	0.00
1400.0	0.8857200E+00	0.8857200E+00	0.7450581E-08	0.00
1600.0	0.8656256E+00	0.8656257E+00	-0.6705523E-07	0.00
1800.0	0.8448796E+00	0.8448795E+00	0.5960464E-07	0.00
2000.0	0.8235680E+00	0.8235678E+00	0.1564622E-06	0.00
5200.0	0.3658517E+00	0.3658517E+00	-0.1117587E-07	0.00
5400.0	0.3477387E+00	0.3477387E+00	0.3725290E-08	0.00
5600.0	0.3301881E+00	0.3301881E+00	-0.7450581E-08	0.00
5800.0	0.3132101E+00	0.3132101E+00	-0.3725290E-07	0.00
7000.0	0.2968122E+00	0.2968123E+00	-0.7823110E-07	0.00
7200.0	0.2809994E+00	0.2809993E+00	0.1192093E-06	0.00
7400.0	0.2657738E+00	0.2657737E+00	0.8940697E-07	0.00
7600.0	0.2511354E+00	0.2511355E+00	-0.1154840E-06	0.00
7800.0	0.2370821E+00	0.2370821E+00	0.4842877E-07	0.00
8000.0	0.2236095E+00	0.2236095E+00	0.4842877E-07	0.00
12200.0	0.5418579E-01	0.5418578E-01	0.1257285E-07	0.00
12400.0	0.5024923E-01	0.5024924E-01	-0.7450581E-08	0.00
12600.0	0.4656935E-01	0.4656935E-01	-0.2793968E-08	0.00
12800.0	0.4313229E-01	0.4313228E-01	0.8567935E-08	0.00
13000.0	0.3992463E-01	0.3992462E-01	0.1396984E-08	0.00
13200.0	0.3693346E-01	0.3693346E-01	0.4656613E-09	0.00
13400.0	0.3414636E-01	0.3414636E-01	0.3725290E-08	0.00
13600.0	0.3155139E-01	0.3155140E-01	-0.7916242E-08	0.00
13800.0	0.2913713E-01	0.2913714E-01	-0.7683411E-08	0.00
14000.0	0.2689264E-01	0.2689264E-01	-0.2561137E-08	0.00
18000.0	0.4882581E-02	0.4882581E-02	0.0000000E+00	0.00
18200.0	0.4463084E-02	0.4463084E-02	0.4074536E-09	0.00
18400.0	0.4078088E-02	0.4078088E-02	-0.4656613E-09	0.00
18600.0	0.3724920E-02	0.3724921E-02	-0.6984919E-09	0.00
18800.0	0.3401096E-02	0.3401096E-02	0.2037268E-09	0.00
19000.0	0.3104309E-02	0.3104309E-02	0.4074536E-09	0.00
19200.0	0.2832422E-02	0.2832422E-02	0.3783498E-09	0.00
19400.0	0.2583452E-02	0.2583452E-02	0.4074536E-09	0.00
19600.0	0.2355563E-02	0.2355563E-02	-0.2910383E-09	0.00
19800.0	0.2147058E-02	0.2147058E-02	-0.2910383E-09	0.00
24000.0	0.2863552E-03	0.2863654E-03	-0.5456968E-10	0.00
24200.0	0.2594256E-03	0.2594255E-03	0.5456968E-10	0.00
24400.0	0.2349661E-03	0.2349662E-03	-0.5456968E-10	0.00
24600.0	0.2127647E-03	0.2127647E-03	-0.4183676E-10	0.00
24800.0	0.1926183E-03	0.1926183E-03	0.2546565E-10	0.00
25000.0	0.1743413E-03	0.1743413E-03	0.3092282E-10	0.00
25200.0	0.1577646E-03	0.1577646E-03	-0.5456968E-10	0.00
25400.0	0.1427337E-03	0.1427337E-03	0.3637979E-10	0.00
25600.0	0.1291078E-03	0.1291079E-03	-0.5820766E-10	0.00
25800.0	0.1167588E-03	0.1167588E-03	0.2455636E-10	0.00
26000.0	0.1055695E-03	0.1055695E-03	0.3910827E-10	0.00

Figure 7-14: Comparison of ADVISER and SENET results for Figure 7-13. Tandem, 2 C and 2 IOL required.

Time	SENET	ADVISER	(SENET-ADVISER)	% Diff.
200.0	0.9998223E+00	0.9999685E+00	-0.1361519E-03	0.01
400.0	0.9992822E+00	0.9998116E+00	-0.5293712E-03	0.05
600.0	0.9983668E+00	0.9995273E+00	-0.1160488E-02	0.12
800.0	0.9970616E+00	0.9990740E+00	-0.2012357E-02	0.20
1000.0	0.9953511E+00	0.9984210E+00	-0.3069885E-02	0.31
1200.0	0.9932191E+00	0.9975388E+00	-0.4319750E-02	0.43
1400.0	0.9906493E+00	0.9963988E+00	-0.5749486E-02	0.58
1600.0	0.9876261E+00	0.9949736E+00	-0.7347405E-02	0.74
1800.0	0.9841342E+00	0.9932366E+00	-0.9102374E-02	0.92
2000.0	0.9801596E+00	0.9911631E+00	-0.1100352E-01	1.12
2200.0	0.9756894E+00	0.9887297E+00	-0.1304028E-01	1.34
2400.0	0.9707126E+00	0.9859143E+00	-0.1520169E-01	1.57
2600.0	0.9652199E+00	0.9826968E+00	-0.1747689E-01	1.81
9200.0	0.5522491E+00	0.6389382E+00	-0.8668911E-01	15.70
9400.0	0.5369108E+00	0.6240145E+00	-0.8710366E-01	16.22
9600.0	0.5216915E+00	0.6090811E+00	-0.8738965E-01	16.75
9800.0	0.5066104E+00	0.5941603E+00	-0.8754990E-01	17.28
10000.0	0.4916856E+00	0.5792730E+00	-0.8758744E-01	17.81
10200.0	0.4769340E+00	0.5644397E+00	-0.8750571E-01	18.35
15800.0	0.1687120E+00	0.2243941E+00	-0.5568206E-01	33.00
16000.0	0.1617050E+00	0.2158566E+00	-0.5415158E-01	33.49
16200.0	0.1549451E+00	0.2075755E+00	-0.5263038E-01	33.97
16400.0	0.1484267E+00	0.1995476E+00	-0.5112094E-01	34.44
16600.0	0.1421441E+00	0.1917695E+00	-0.4962545E-01	34.91
21800.0	0.4279192E-01	0.6216241E-01	-0.1937049E-01	45.27
22000.0	0.4076578E-01	0.5935029E-01	-0.1858451E-01	45.59
22200.0	0.3883022E-01	0.5665510E-01	-0.1782488E-01	45.90
22400.0	0.3698158E-01	0.5407268E-01	-0.1709110E-01	46.22
22600.0	0.3521631E-01	0.5159896E-01	-0.1638265E-01	46.52
25200.0	0.1844592E-01	0.2766751E-01	-0.9221594E-02	49.99
25400.0	0.1753758E-01	0.2634561E-01	-0.8808033E-02	50.22
25600.0	0.1667235E-01	0.2508355E-01	-0.8411204E-02	50.45
25800.0	0.1584828E-01	0.2387885E-01	-0.8030573E-02	50.67
26000.0	0.1506352E-01	0.2272911E-01	-0.7665589E-02	50.89
28800.0	0.7331980E-02	0.1125228E-01	-0.3920299E-02	53.47
29000.0	0.6960301E-02	0.1069262E-01	-0.3732322E-02	53.62
29200.0	0.6606983E-02	0.1015982E-01	-0.3552836E-02	53.77
29400.0	0.6271152E-02	0.9652642E-02	-0.3381490E-02	53.92
29600.0	0.5951971E-02	0.9169924E-02	-0.3217953E-02	54.07
32400.0	0.2844847E-02	0.4430988E-02	-0.1586141E-02	55.75
32600.0	0.2697490E-02	0.4204167E-02	-0.1506677E-02	55.85
32800.0	0.2557619E-02	0.3988665E-02	-0.1431046E-02	55.95
33000.0	0.2424864E-02	0.3783937E-02	-0.1359073E-02	56.05
33200.0	0.2298872E-02	0.3589462E-02	-0.1290590E-02	56.14
33400.0	0.2179307E-02	0.3404743E-02	-0.1225436E-02	56.23
39000.0	0.4787798E-03	0.7565972E-03	-0.2778174E-03	58.03
39200.0	0.4532638E-03	0.7164731E-03	-0.2632093E-03	58.07
39400.0	0.4290907E-03	0.6784452E-03	-0.2493545E-03	58.11
39600.0	0.4061908E-03	0.6424061E-03	-0.2362153E-03	58.15
39800.0	0.3844980E-03	0.6082538E-03	-0.2237558E-03	58.19
40000.0	0.3639497E-03	0.5758913E-03	-0.2119416E-03	58.23

Figure 7-15: Comparison of ADVISER and SENET results for Figure 7-13, Tandem, 1 C and 1 IOL required.

to result from possible slight differences in the ADVISER reliability model of the architecture as compared to the model assumed in the SENET program, especially for low values of the requirements (see [Siewiorek 78], Page III-76).

7.3.4 The Global Bus architecture

The next example from SENET used as a test of ADVISER concerns the Global Bus multiprocessor architecture. Two cases of the PMS diagram of the architecture used in the test are shown in Figure 7-16. The Global Bus architecture is similar to the architecture of a single cluster of Cm* described in Section 7.3.1. The global bus shared by all the processors is analogous to the Kmap of a Cm* cluster. Each of the processors is able to access memories in other clusters via the global bus and the failure of a processor does not preclude accessing of its associated memories by processors in other clusters.

The attempts at matching results for the Global Bus architecture without I/O lines (Figure 7-16(a)) were successful for all requirements for low values of the number of memories configured per processor[38]. However, discrepancies appeared when the number of memories per processor exceeded three. All attempts to obtain a match with SENET results failed in the case of the Global Bus architecture with I/O lines (Figure 7-16(b)). Subsequently, manual construction was used to verify the correctness of ADVISER output in both cases of Figure 7-16. Figure 7-17 shows the comparison of SENET and ADVISER results for the case of Figure 7-16(b) with a requirement of "ψ(2,P) $\wedge$ ψ(8,M) $\wedge$ ψ(1,IOL)". The mismatch is obvious. The reliability for this example was derived by hand, using the success-state-table method described in SENET, to be

$$R_{sys} = R_{bus} * R_{S.IO} * R_{IOL} * (R_0 + 2R_1 + 3R_2 + 6R_3 + R_4 + 2R_5 + 2R_6)$$

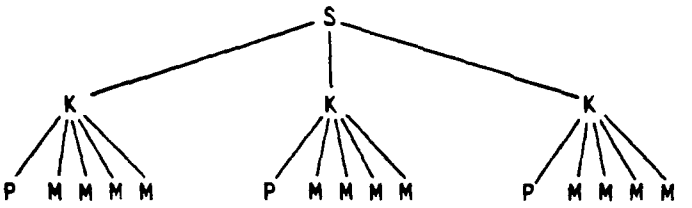
where

$$R_0 = R_{K.IO}^2 * R_K^3 * R_P^3 * R_{AGM}$$

$$R_1 = R_{K.IO}(1-R_{K.IO}) * R_K^3 * R_P^3 * R_{AGM}$$

---

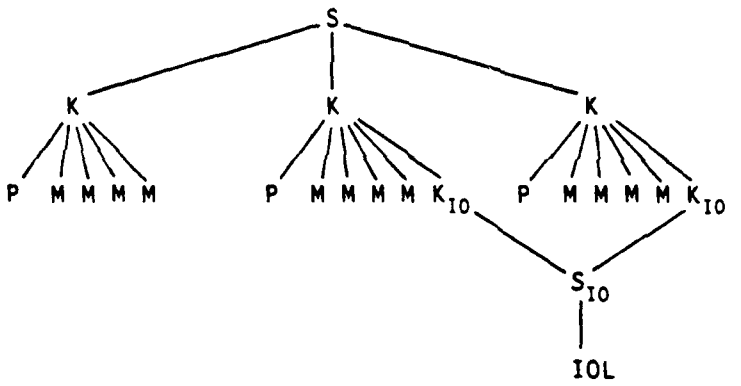
[38] Note here that the deficiency in the TREEREL algorithm which caused ADVISER to compute a different reliability for the Cm* structure for small values of the requirements does not have adverse effects in the Global Bus case since the bus was forced to be in the Kernel (by the method described in Section 7.3.1) and the bus controller K for a processor is always necessary in any system state in which the processor is functional.



Key

S	Global Bus	P	Processor
K	Bus Control	M	Memory

(a)



Key

S	Global Bus	P	Processor	K_{IO}	I/O Interface
K	Bus Control	M	Memory	S_{IO}	I/O Switch
		IOL	I/O Line		

(b)

Figure 7-16: The Global Bus architecture used for ADVISER tests
(a) Without I/O lines (b) With I/O line.

Time	SENET	ADVISER	(SENET-ADVISER)	% Diff.
200.0	0.9877112E+00	0.9886170E+00	-0.9058192E-03	0.09
400.0	0.9728974E+00	0.9683741E+00	0.4523285E-02	0.46
600.0	0.9560773E+00	0.9418494E+00	0.1442794E-01	1.51
800.0	0.9376178E+00	0.9102775E+00	0.2734021E-01	2.92
1000.0	0.9177602E+00	0.8756564E+00	0.4210385E-01	4.59
1200.0	0.8966526E+00	0.8388458E+00	0.5780577E-01	6.45
1400.0	0.8743814E+00	0.8006632E+00	0.7373115E-01	8.43
1600.0	0.8509986E+00	0.7616787E+00	0.8932012E-01	10.50
1800.0	0.8265441E+00	0.7224027E+00	0.1041414E+00	12.60
2000.0	0.8010601E+00	0.6831903E+00	0.1178698E+00	14.71
2200.0	0.7746032E+00	0.6443451E+00	0.1302581E+00	16.82
2400.0	0.7472490E+00	0.6060753E+00	0.1411737E+00	18.89
2600.0	0.7190950E+00	0.5686083E+00	0.1504867E+00	20.93
2800.0	0.6902589E+00	0.5320988E+00	0.1581601E+00	22.91
3000.0	0.6608755E+00	0.4966852E+00	0.1641913E+00	24.84
3200.0	0.6310974E+00	0.4624828E+00	0.1686146E+00	26.72
3400.0	0.6010805E+00	0.4295866E+00	0.1714939E+00	28.53
3600.0	0.5709892E+00	0.3980729E+00	0.1729163E+00	30.28
3800.0	0.5409868E+00	0.3680003E+00	0.1729865E+00	31.98
4000.0	0.5112325E+00	0.3394101E+00	0.1718224E+00	33.61
4200.0	0.4818778E+00	0.3123287E+00	0.1695491E+00	35.19
4400.0	0.4530637E+00	0.2867671E+00	0.1662966E+00	36.70
4600.0	0.4249180E+00	0.2627229E+00	0.1621951E+00	38.17
4800.0	0.3975545E+00	0.2401615E+00	0.1573930E+00	39.59
5000.0	0.3710715E+00	0.2191171E+00	0.1519544E+00	40.95
13600.0	0.4246648E-02	0.1120705E-02	0.3125943E-02	73.61
13800.0	0.3742598E-02	0.9717571E-03	0.2770841E-02	74.04
14000.0	0.3296368E-02	0.8421255E-03	0.2454243E-02	74.45
14200.0	0.2901539E-02	0.7293818E-03	0.2172157E-02	74.86
14400.0	0.2552465E-02	0.6313906E-03	0.1921074E-02	75.26
14600.0	0.2244070E-02	0.5462750E-03	0.1697794E-02	75.66
14800.0	0.1971804E-02	0.4723922E-03	0.1499412E-02	76.04
15000.0	0.1731604E-02	0.4082963E-03	0.1323308E-02	76.42
15200.0	0.1519836E-02	0.3527244E-03	0.1167112E-02	76.79
15400.0	0.1333259E-02	0.3045707E-03	0.1028688E-02	77.16
15600.0	0.1168974E-02	0.2628684E-03	0.9061056E-03	77.51
15800.0	0.1024418E-02	0.2267729E-03	0.7976451E-03	77.86
16000.0	0.8972958E-03	0.1955469E-03	0.7017489E-03	78.21
16200.0	0.7855669E-03	0.1685475E-03	0.6170194E-03	78.54
16400.0	0.6874278E-03	0.1452145E-03	0.5422133E-03	78.88
16600.0	0.6012693E-03	0.1250598E-03	0.4762095E-03	79.20
16800.0	0.5256757E-03	0.1076580E-03	0.4180167E-03	79.52
17000.0	0.4593804E-03	0.9264281E-04	0.3667376E-03	79.83
17200.0	0.4012734E-03	0.7969032E-04	0.3215831E-03	80.14
17400.0	0.3503785E-03	0.6852291E-04	0.2818556E-03	80.44
17600.0	0.3058016E-03	0.5889875E-04	0.2469029E-03	80.74
17800.0	0.2667978E-03	0.5060813E-04	0.2161897E-03	81.03
18000.0	0.2326742E-03	0.4346919E-04	0.1892050E-03	81.32
18200.0	0.2028346E-03	0.3732446E-04	0.1655101E-03	81.60
18400.0	0.1767576E-03	0.3203756E-04	0.1447200E-03	81.87
18600.0	0.1539811E-03	0.2749050E-04	0.1264906E-03	82.15
18800.0	0.1340881E-03	0.2358121E-04	0.1105069E-03	82.41
19000.0	0.1167208E-03	0.2022148E-04	0.9649932E-04	82.68
19200.0	0.1015738E-03	0.1733510E-04	0.8423870E-04	82.93

Figure 7-17: Comparison of ADVISER and SENET results for Figure 7-16. Global Bus, 2 P, 8 M and 1 IOL required.

$$R_2 = R_{K.IO}^2 \cdot R_K^3 \cdot R_P^2(1-R_P) \cdot R_{AGM}$$

$$R_3 = R_{K.IO}(1-R_{K.IO}) \cdot R_K^3 \cdot R_P^2(1-R_P) \cdot R_{AGM}$$

$$R_4 = R_{K.IO}^2 \cdot R_K^2(1-R_K) \cdot R_P^2 \cdot R_M^8$$

$$R_5 = R_{K.IO}(1-R_{K.IO}) \cdot R_K^2(1-R_K) \cdot R_P^2 \cdot R_M^8$$

$$R_6 = R_{K.IO} \cdot R_K^2(1-R_K) \cdot R_P^2 \cdot R_M^8$$

and

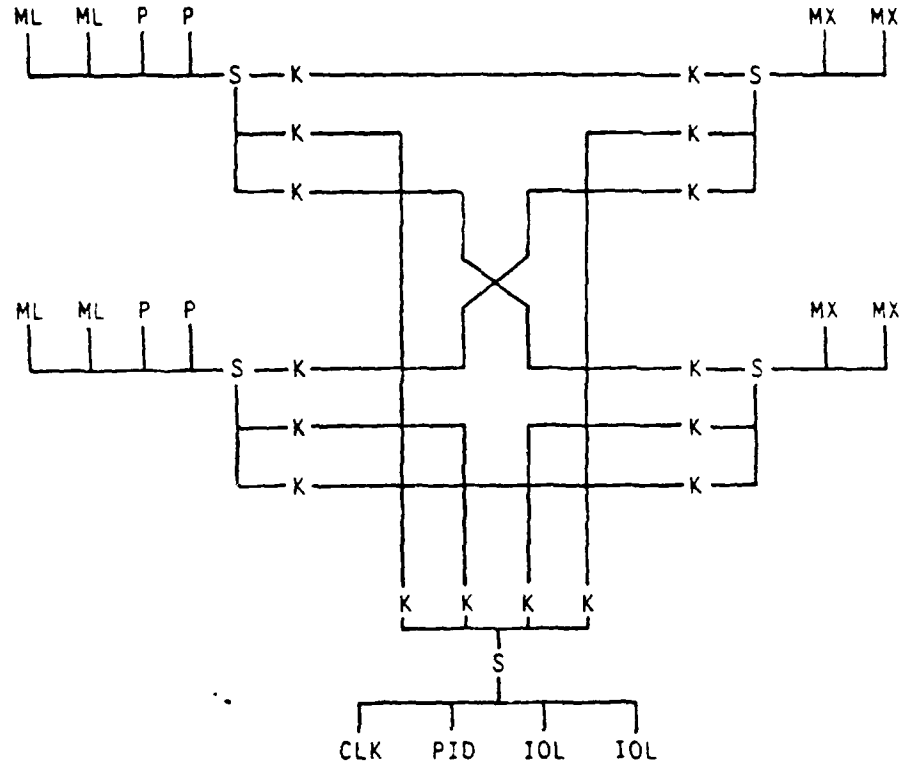
$$R_{AGM} = \Sigma_{i=0}^{4} \binom{12}{i} R_M^{12-i} (1-R_M)^i$$

This reliability was compared to the output of ADVISER after substitution of R_0 through R_6 in the R_{sys} polynomial by using MACSYMA as described in Listing 5 on Page 196. The subtraction of the two polynomials resulted identically in zero, thereby showing that the ADVISER output was correct for this case. A further test was made using a simpler version of the architecture with only two memories per processor and the requirement "$\psi(2,P) \wedge \psi(2,M) \wedge \psi(1,IOL)$". The reliability for this simpler case was computed with ADVISER as well as by the LISP program of Section 7.2 working on a hand-constructed RBD for the case. When symbolically compared with MACSYMA these two reliabilities were found to be identical.

7.3.5 The Pluribus architecture

The last of the five examples from SENET which were used to test ADVISER is the Pluribus architecture which is described in [Ornstein 75]. The version of the architecture used to test ADVISER is shown in Figure 7-18. Each of the processor buses has one or more processors (P in the figure) and at least one local memory (ML) per processor has to function if a processor bus is to be functional. On the memory buses are situated the shared memories (MX) accessible by any processor on a processor bus. The I/O bus (there may be more than one) connects to the I/O devices which are also accessible by any processor. The system clock (CLK in the figure) and the Pseudo Interrupt Device (PID in the figure) are both essential for system functioning and also reside on the I/O bus.

For the test the architecture used had two processors and two local memories on each of



Key

P	Processor	K	Bus Coupler
ML	Local Memory	CLK	Clock
MX	Shared Memory	PID	Pseudo-Interrupt Device
S	Bus	IOL	I/O Line

Figure 7-18: Pluribus model for ADVISER test.

Time	SENET	ADVISER	(SENET-ADVISER)	% Diff
200.0	0.9945158E+00	0.9948578E+00	-0.3420487E-03	0.03
400.0	0.9881578E+00	0.9881006E+00	0.5715340E-04	0.01
600.0	0.9809476E+00	0.9797324E+00	0.1215190E-02	0.12
800.0	0.9729111E+00	0.9697837E+00	0.3127404E-02	0.32
1000.0	0.9640771E+00	0.9583082E+00	0.5768895E-02	0.60
1200.0	0.9544773E+00	0.9453768E+00	0.9100489E-02	0.95
1400.0	0.9441460E+00	0.9310750E+00	0.1307096E-01	1.38
1600.0	0.9331194E+00	0.9154986E+00	0.1762076E-01	1.89
1800.0	0.9214355E+00	0.8987511E+00	0.2268440E-01	2.46
2000.0	0.9091336E+00	0.8809416E+00	0.2819197E-01	3.10
2200.0	0.8962542E+00	0.8621820E+00	0.3407221E-01	3.80
2400.0	0.8828383E+00	0.8425849E+00	0.4025344E-01	4.56
2600.0	0.8689275E+00	0.8222626E+00	0.4666494E-01	5.37
2800.0	0.8545635E+00	0.8013255E+00	0.5323795E-01	6.23
10200.0	0.2989771E+00	0.1621089E+00	0.1368682E+00	45.78
10400.0	0.2880181E+00	0.1535380E+00	0.1344801E+00	46.69
10600.0	0.2773625E+00	0.1453676E+00	0.1319949E+00	47.59
10800.0	0.2670085E+00	0.1375836E+00	0.1294249E+00	48.47
11000.0	0.2569537E+00	0.1301721E+00	0.1267816E+00	49.34
11200.0	0.2471951E+00	0.1231193E+00	0.1240758E+00	50.19
11400.0	0.2377295E+00	0.1164113E+00	0.1213182E+00	51.03
11600.0	0.2285532E+00	0.1100348E+00	0.1185184E+00	51.86
11800.0	0.2196622E+00	0.1039763E+00	0.1156859E+00	52.67
12000.0	0.2110522E+00	0.9822276E-01	0.1128294E+00	53.46
12200.0	0.2027185E+00	0.9276142E-01	0.1099571E+00	54.24
12400.0	0.1946564E+00	0.8757975E-01	0.1070767E+00	55.01
12600.0	0.1868607E+00	0.8266558E-01	0.1041951E+00	55.76
12800.0	0.1793263E+00	0.7800705E-01	0.1013193E+00	56.50
27400.0	0.5390093E-02	0.7767298E-03	0.4613363E-02	85.59
27600.0	0.5115320E-02	0.7273190E-03	0.4388001E-02	85.78
27800.0	0.4854218E-02	0.6810305E-03	0.4173187E-02	85.97
28000.0	0.4606130E-02	0.6376685E-03	0.3968461E-02	86.16
28200.0	0.4370430E-02	0.5970500E-03	0.3773380E-02	86.34
28400.0	0.4146518E-02	0.5590028E-03	0.3587515E-02	86.52
28600.0	0.3933826E-02	0.5233657E-03	0.3410460E-02	86.70
28800.0	0.3731808E-02	0.4899871E-03	0.3241821E-02	86.87
29000.0	0.3539945E-02	0.4587251E-03	0.3081220E-02	87.04
29200.0	0.3357742E-02	0.4294468E-03	0.2928295E-02	87.21
29400.0	0.3184727E-02	0.4020270E-03	0.2782700E-02	87.38
29600.0	0.3020450E-02	0.3763489E-03	0.2644101E-02	87.54
29800.0	0.2864483E-02	0.3523025E-03	0.2512181E-02	87.70
30000.0	0.2716416E-02	0.3297848E-03	0.2386631E-02	87.86
37400.0	0.3694320E-03	0.2829785E-04	0.3411341E-03	92.34
37600.0	0.3498083E-03	0.2647468E-04	0.3233336E-03	92.43
37800.0	0.3312183E-03	0.2476876E-04	0.3064495E-03	92.52
38000.0	0.3136080E-03	0.2317258E-04	0.2904354E-03	92.61
38200.0	0.2969265E-03	0.2167909E-04	0.2752474E-03	92.70
38400.0	0.2811252E-03	0.2028169E-04	0.2608435E-03	92.79
38600.0	0.2661584E-03	0.1897423E-04	0.2471842E-03	92.87
38800.0	0.2519823E-03	0.1775092E-04	0.2342314E-03	92.96
39000.0	0.2385556E-03	0.1660636E-04	0.2219492E-03	93.04
39200.0	0.2258392E-03	0.1553549E-04	0.2103037E-03	93.12
39400.0	0.2137958E-03	0.1453358E-04	0.1992622E-03	93.20
39600.0	0.2023902E-03	0.1359619E-04	0.1887940E-03	93.28
39800.0	0.1915889E-03	0.1271918E-04	0.1788697E-03	93.36
40000.0	0.1813601E-03	0.1189866E-04	0.1694614E-03	93.44

Figure 7-19: Comparison of ADVISER and SENET results for Figure 7-18, Pluribus, 2 P, 2 MX, 2 ML, 1 CLK, 1 PID, 1 IOL

two processor buses, two shared memories on each of two memory buses and two I/O lines, a clock and a PID on the single I/O bus. The requirements used were "$\psi(2,P) \wedge \psi(2,MX) \wedge \psi(2,ML) \wedge \psi(1,CLK) \wedge \psi(1,PID) \wedge \psi(1,IOL)$". The results are shown in Figure 7-19. Again the mismatch is unmistakable.

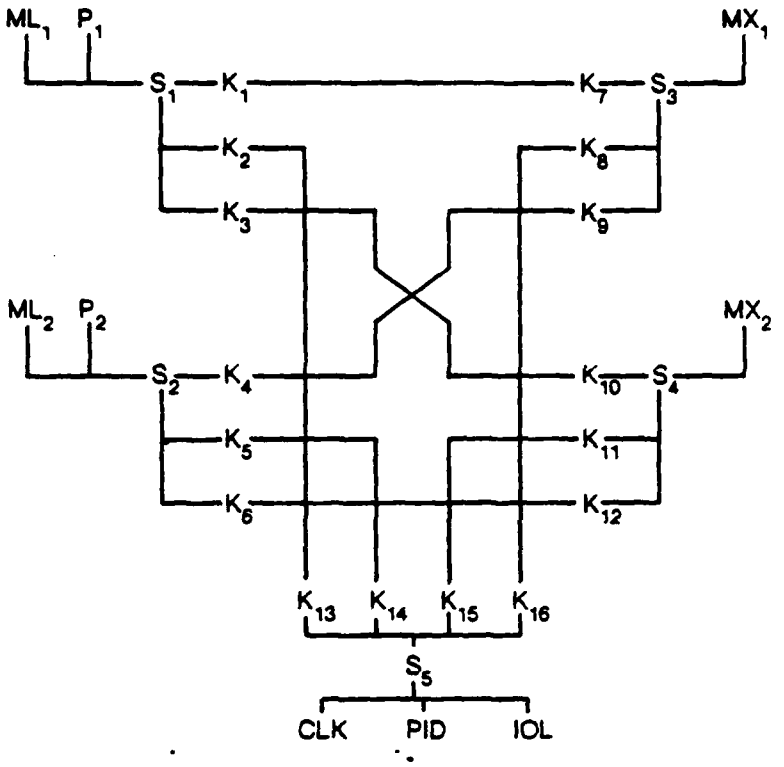
This mismatch, as in the case of the Cm* mismatches, was anticipated since it arose from a known deficiency of ADVISER. This deficiency was the subject of Chapter 6, Section 6.9.3 which considered the side constraint dealing with bounded-clustering of critical components. The deficiency is summarized here. In order to implement a complete form of the bounded-clustering side constraint it is necessary to process a set of general inequalities which relate the quantities of the different types of critical components to be chosen in a bounded cluster. The current version of ADVISER allows only a weaker set of inequalities to be specified which simply put an independent integer lower-bound on the numbers of each type of critical component to be chosen in a bounded cluster. Relating this to the Pluribus example under study, the number of local memories which need to be functional in order to insure that a processor bus functions is bounded on the lower side by the *number of processors which are functional on that bus*. Thus the bounded-cluster constraint should be

$$(\text{Number of ML}) \geq (\text{Number of P}) \wedge (\text{Number of P}) \geq 1$$

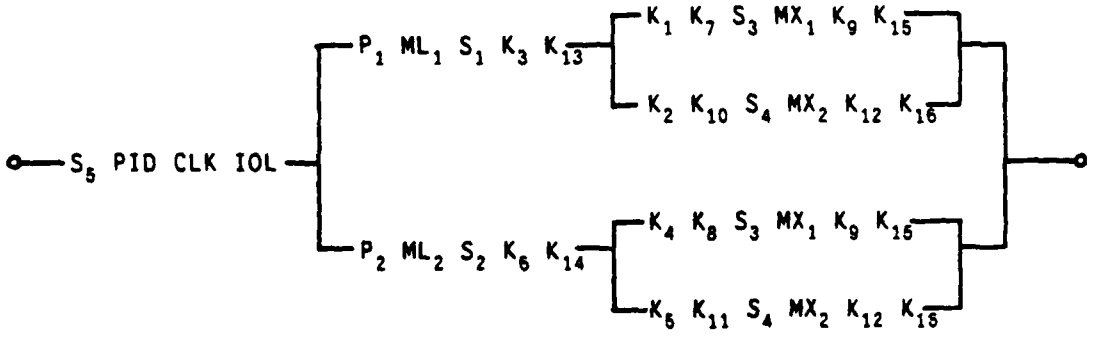
instead of which it was only possible to give ADVISER the weaker constraint

$$(\text{Number of ML}) \geq 1 \wedge (\text{Number of P}) \geq 1$$

In effect, therefore, the mismatch in Figure 7-19 reflects the fact that the SENET program for the architecture and ADVISER were employing slightly different models of the architecture. The question now arises, if one is to grant the correctness of the model assumed by ADVISER, is the reliability it computes correct for that model? It was decided to investigate this by comparing to hand-computation. However, due to the complexity of the architecture and the inability of the LISP program to cope with the size of the hand-constructed RBD for Figure 7-18 it was decided to use a simpler form of the architecture. The example actually used for comparison (Figure 7-20(a)) was similar to that of Figure 7-18 except that there was only one processor and a single local memory on each of two processor buses, one shared memory on each of two memory buses, and one I/O line, one clock and one PID on the single I/O bus. The requirement used was "$\psi(1,P) \wedge \psi(1,ML) \wedge \psi(1,MX) \wedge \psi(1,CLK) \wedge \psi(1,PID) \wedge \psi(1,IOL)$". Figure 7-20(b) shows the hand-constructed RBD for the simpler example with this requirement. Listing 8 shows the MACSYMA terminal session which demonstrated that the two symbolic expressions were the same.



(a)



(b)

Figure 7-20: (a) Simple version of Pluribus architecture (b) Hand-constructed SPRBD for structure in (a) above; 1 P, 1 ML, 1 MX, 1 CLK, 1 PID and 1 IOL required.

Listing 8, Simple Pluribus Example

This listing shows the MACSYMA session during which ADVISER output for the simple Pluribus example was shown to be equal to the manual construction.

```
*:a

This is MACSYMA 294

FIX294 2 DSK MACSYM being loaded
Loading done

(C1) batch(kini,egadv);

(C2) /*
-----------------------------------------------------------------------
MACSYMA Module for Reliability Function manipulation
   produced by ADVISER on Saturday, 17 Jan 81 at 21:52:35 for [4,1367]
-----------------------------------------------------------------------
Task Title: PLURI.PMS --- A simple 2-PcBus, 2-MBus, 1-IOBus Pluribus model


Requirements on the Structure were:

(1-OF-P AND 1-OF-ML AND 1-OF-MX AND 1-OF-PID AND 1-OF-CLK AND 1-OF-IOI)

Component-Type definitions for this task:

INDEX  TYPENAME     PRINTNAME   REL.FN.  PARAMS
-----  --------     ---------   -------  ------
    0  BUS          BUS         Const.   Relia.=1.00000000
    1  COUPLER      K           Expon.   Lambda=27.00000000
    2  MEXT         MX          Expon.   Lambda=23.13900000
    3  MLOCAL       ML          Const.   Relia.=23.13900000
    4  ARBITERP     AP          Expon.   Lambda=3.50000000
    5  ARBITERM     AM          Expon.   Lambda=3.50000000
    6  PROCESSOR    P           Expon.   Lambda=12.90200000
    7  IOINT        IOI         Expon.   Lambda=72.20000000
    8  CLOCK        CLK         Expon.   Lambda=14.70400000
    9  PID          PID         Expon.   Lambda=7.00000000

PMS Structure Definitions for this  task:
```

```
INDEX  NAME       TYPE         NNEIG  NEIGHBORS
-----  ----       ----         -----  ---------
    0  SM.1       BUS              4  (AM.1, K.1, K.2, K.3)
    1  SM.2       BUS              4  (AM.2, K.4, K.5, K.6)
    2  SP.1       BUS              4  (AP.1, K.7, K.8, K.9)
    3  SP.2       BUS              4  (AP.2, K.10, K.11, K.12)
    4  SIO        BUS              7  (T.1, PID.1, CLK.1, K.13, K.14,
                                       K.15, K.16)
    5  AM.1       ARBITERM         2  (SM.1, MX.1)
    6  AM.2       ARBITERM         2  (SM.2, MX.3)
    7  AP.1       ARBITERP         3  (SP.1, P.1, ML.1)
    8  AP.2       ARBITERP         3  (SP.2, P.3, ML.3)
    9  MX.1       MEXT             1  (AM.1)
   10  MX.3       MEXT             1  (AM.2)
   11  P.1        PROCESSOR        1  (AP.1)
   12  P.3        PROCESSOR        1  (AP.2)
   13  ML.1       MLOCAL           1  (AP.1)
   14  ML.3       MLOCAL           1  (AP.2)
   15  T.1        IOINT            1  (SIO)
   16  PID.1      PID              1  (SIO)
   17  CLK.1      CLOCK            1  (SIO)
   18  K.1        COUPLER          2  (SM.1, K.7)
   19  K.2        COUPLER          2  (SM.1, K.10)
   20  K.3        COUPLER          2  (SM.1, K.14)
   21  K.4        COUPLER          2  (SM.2, K.8)
   22  K.5        COUPLER          2  (SM.2, K.11)
   23  K.6        COUPLER          2  (SM.2, K.13)
   24  K.7        COUPLER          2  (SP.1, K.1)
   25  K.8        COUPLER          2  (SP.1, K.4)
   26  K.9        COUPLER          2  (SP.1, K.16)
   27  K.10       COUPLER          2  (SP.2, K.2)
   28  K.11       COUPLER          2  (SP.2, K.5)
   29  K.12       COUPLER          2  (SP.2, K.15)
   30  K.13       COUPLER          2  (SIO, K.6)
   31  K.14       COUPLER          2  (SIO, K.3)
   32  K.15       COUPLER          2  (SIO, K.12)
   33  K.16       COUPLER          2  (SIO, K.9)

------------------------------------------------------------------------
*/

%%T0:
        BUS * MX * AM;
(D2)                              AM BUS MX

(C3) %%T6:
        BUS * ML * AP * P;
(D3)                            AP BUS ML P

(C4) %%T8:
        BUS * IOI * CLK * PID;
(D4)                          BUS CLK IOI PID

(C5) /* End of temporary variable initializations*/

System%Reliability: 0;
(D5)                                  0

(C6) System%Reliability:
        4 * K↑6 * %%T0 * %%T6 * %%T8  -  2 * K↑10 * %%T0 * %%T6↑2 *
        %%T8  -  2 * K↑10 * %%T0↑2 * %%T6 * %%T8  -  2 * K↑12 * %%T0↑2
         * %%T6↑2 * %%T8  +  4 * K↑14 * %%T0↑2 * %%T6↑2 * %%T8  -
        K↑16 * %%T0↑2 * %%T6↑2 * %%T8
        ;
```

```
         2   2    5          16   2   2  2
(D6) - AM  AP  BUS  CLK IOI K   ML  MX  P  PID

       2   2    5          14   2   2  2
 + 4 AM  AP  BUS  CLK IOI K   ML  MX  P  PID

       2   2    6          12   2   2  2
 - 2 AM  AP  BUS  CLK IOI K   ML  MX  P  PID

         2   4          10   2      2
 - 2 AM AP  BUS  CLK IOI K   ML  MX P  PID

       2       4          10      2
 - 2 AM  AP BUS  CLK IOI K   ML MX  P PID

             3          6
 + 4 AM AP BUS  CLK IOI K  ML MX P PID

(C7) /*End of System Reliability computation*/

FACTOR(%);
                3          6                    2  10
(D7) - AM AP BUS  CLK IOI K  ML MX P (AM AP BUS  K   ML MX P

             2  8                      2  6                      4
 - 4 AM AP BUS  K  ML MX P + 2 AM AP BUS  K  ML MX P + 2 AP BUS K  ML P

            4
 + 2 AM BUS K  MX - 4) PID

(D8)                                BATCH DONE

(C9) batch(kini,eghnd);

(C10) /* Reliability Function printed by LISP at 17-Jan-81 21:48:10 */

SYSREL:
+4*AM*AP*BUS-3*CLK*IOI*K+6*ML*MX*P*PID-2*AM*AP+2*BUS+4*CLK*IOI*K-10*ML-2*MX*P+2
*PID-2*AM+2*AP*BUS+4*CLK*IOI*K+10*ML*MX+2*P*PID
-2*AM+2*AP+2*BUS+5*CLK*IOI*K+12*ML+2*MX+2*P+2*PID+4*AM+2*AP+2*BUS+5*CLK*IOI*K+
14*ML+2*MX+2*P+2*PID-1*AM+2*AP+2*BUS+5*CLK*IOI*K+16*ML+2*MX+2*P+2*PID,

          2   2    5          16   2   2  2
(D10) - AM  AP  BUS  CLK IOI K   ML  MX  P  PID

       2   2    5          14   2   2  2
 + 4 AM  AP  BUS  CLK IOI K   ML  MX  P  PID

       2   2    5          12   2   2  2
 - 2 AM  AP  BUS  CLK IOI K   ML  MX  P  PID

         2   4          10   2      2
 - 2  M AP  BUS  CLK IOI K   ML  MX P  PID

       2       4          10      2
 - 2 AM  AP BUS  CLK IOI K   ML MX  P PID

             3          6
 + 4 AM AP BUS  CLK IOI K  ML MX P PID

(D11)                               BATCH DONE

(C11) ratexpand('sysrel - system%reliability),
(D11)                                        0

(C12) quit();
```

:KILL
•

7.4 Performance measurements on ADVISER

The author's experience with the current version of ADVISER has generated some intuitions regarding the capabilities of the program in terms of the size of problems it can handle. This section describes some of those intuitions and presents some timing measurements made on ADVISER using the Cm* architecture. *Note: The timings in this section are to be used as rough guides to the performance. They are not very accurate due to unavoidable circumstances at the time of measurement. The run time totals include a small part of the timing overhead. All timings are in seconds of CPU time on a Digital Equipment Corp. KL-10 processor.*

The Cm* architecture was chosen because it embodies attributes which cause the current version of ADVISER to exhibit some of its worst case behavior. Although ADVISER does make use of some of the symmetry in the PMS structure there is room for it to do more. The Cm* case offers an example of a structure with *Pendant Tree Subgraphs* which in addition to being symmetric to each other also have considerable symmetry within themselves. Assume that a single Cm* cluster has a total of 12 memories within it and the requirement is for six of them to function. Currently ADVISER does not make use of the fact that the structure of the cluster itself is very regular. It treats the problem as a 6-out-of-12 structure and goes through considerable computation and use of the PMERGE and SMERGE algorithms to generate the canonical-form solution whereas one would straightforwardly write down the expression

$$R_{Kmap} * R_{Slocal} * \Sigma_{i=0}^{6} \binom{12}{i} R_M^{12-i} (1-R_M)^i \qquad (7.1)$$

Note, however, that in a situation where this is a small portion of a bigger problem, the expression (7.1) does not retain any of the individual identities of the memories involved and therefore computing the symbolic probability of the intersection or union of this event with other events dependent on it would not be easy. Furthermore, the TREEREL algorithm in ADVISER works even when no symmetry exists although the price for this generality is paid when there is a lot of symmetry internal to the PTSs which could be exploited.

The above example was provided to indicate that such problems would cause ADVISER to do large amounts of computation especially if the requirement asks for roughly half of the

configured components of each critical type to be functional. This is where the binomial distribution of the number of functional cases reaches its peak. Experiments also provided the intuition that PMS structures which have smaller Kernels for a given set of PTSs tend to do better in terms of compute time consumed. Experimental results given below show that the CRP algorithms are the biggest performance bottleneck. Kernel path CRPs are used directly in the collapsing of the CRPTree, thus performance deteriorates when there are more of them and they have more terms. The effect of increasing the number of PTSs in the PMS structure is not as drastic in comparison, especially if the PTSs are symmetric. This is because CRPs for the pendant tree subgraphs are allotted unique bits in the AUXVEC bitvector and the number of them and their lengths do not have a significant effect in the CRPTree collapsing process. Increasing the *complexity* of the individual PTSs in a symmetric set, therefore, has a greater effect than increasing the *number* of them in the set. However, a larger number of PTSs implies a larger number of segments of the PMS graph which implies an increased the depth to the CRPTree. In turn, the number of compositions of the requirement integers in the Compositions Table grows combinatorially with the number of segments. It is experimentally observed that these two properties combine to have a significant adverse effect on the run time after approximately six or eight segments have been introduced.

It appears that although compute time is necessarily sensitive to the number of components in the structure *per se*, it is more sensitive to how many of the configured critical components are required to be functional i.e. the complexity of the requirements (see Footnote on Page 109). ADVISER does best when the requirements demand that much less than half, or almost all, of the configured components of each critical type be functional. This is generally expected to be the case in practical multiprocessor architectures. Tasks on multiprocessors such as Cm* will probably require much fewer than half the total number of processors configured. Likewise array-processor type SIMD archictectures will generally require all, or almost all processors to be functional.

It is to be noted that all of the above patterns of behavior do not in themselves affect the compute time significantly: their effect is magnified by the poor performance of the CRP algorithms. Experiments show that the outstanding consumers of computation time during a typical ADVISER run are the CRP merging algorithms PMERGE and SMERGE. The timing measurements presented below show that overwhelming percentages of the run time are spent in these algorithms despite all efforts to make them as efficient as possible in the implementation. This is not entirely unexpected since dealing with canonical forms of the polynomials is known to be inefficient in a larger sense. On the other hand the versatility, simplicity and robustness of the PMERGE and SMERGE algorithms greatly eased the

implementation of the program. It is clear that for more efficient versions of ADVISER some more compact low level representation for the polynomials would have to be found which allows greater use of existing symmetry in the structure while preserving the ease with which the symbolic probabilities of the unions and intersections of dependent events may be computed.

Requirements		Total Run Time (sec)	Total Time in Merge Package (sec)	% Merge/ Total	Total No. CRP Terms processed
P	M				
1	1	1.41	0.427	30.28	198
	2	1.89	0.776	41.06	581
	3	2.59	1.255	48.46	1071
	4	2.84	1.382	48.66	1353
	5	2.97	1.440	48.48	1427
	6	3.08	1.490	48.38	1439
2	1	1.32	0.293	22.20	225
	2	1.84	0.684	37.17	608
	3	2.47	1.146	46.40	1098
	4	2.91	1.388	47.70	1380
	5	3.08	1.487	48.28	1454
	6	3.21	1.582	49.28	1466
3	1	1.26	0.189	15.00	234
	2	1.74	0.574	32.99	617
	3	2.46	1.114	45.28	1107
	4	2.93	1.429	48.77	1389
	5	3.21	1.554	48.41	1463
	6	3.43	1.678	48.92	1475

Table 7-1: ADVISER timings for 1-cluster Cm* case.

Two simple versions of the Cm* architecture were used to generate what is conceivably worst case behavior for ADVISER. The first example is a simple single-cluster Cm* with three Cm's each with one processor and two memories. This makes a total of three processors and six memories. All possible combinations of requirements on processors and memories were taken. The results are shown in Table 7-1. The first column gives the total run time for each case. The second column gives the run time consumed by the PMERGE and SMERGE algorithms for each case. The third column states the merge run time as a percentage of the total run time. The last column shows the total number of CRP terms processed in the intermediate representation package for each case.

The second example was a two cluster model wherein each cluster was identical to the cluster in the first case and the two clusters were connected through a single intercluster bus. This makes a total of six processors and twelve memories. Enough cases of requirements

Requirements		Total Run Time (sec)	Total Time in Merge Package (sec)	% Merge/ Total	Total No. CRP Terms processed
P	M				
1	1	5.20	0.903	17.36	418
	2	48.91	42.449	86.78	2520
	3	107.02	94.469	88.27	5500
	4	147.04	130.230	88.56	7970
	5	163.26	143.802	88.08	9504
	6	166.56	143.773	86.32	10680
	7	158.27	139.248	87.98	10412
	8	64.66	55.504	85.84	6804
2	1	7.00	1.984	28.34	926
	2	77.83	60.314	77.43	4048
	3	166.39	133.686	80.34	9016
	4	225.85	183.160	81.10	13322
	5	254.53	202.315	79.48	16242
	6	>244.00	?	?	?
	7	241.55	198.788	82.29	16492
	8	95.16	77.496	81.44	9936
3	1	4.34	1.868	43.04	1648
	2	70.80	59.662	84.27	6274
	3	160.28	137.367	85.70	12944
	4	226.86	190.701	84.05	19292
	5	>253.00	?	?	?
	6	479.61	326.610	68.10	27572
	7	284.89	217.165	76.23	22024
	8	107.52	83.091	77.28	12674
4	1	3.61	1.670	46.26	1236
	2	64.89	55.658	85.10	4862
	3	144.82	126.736	87.51	10032
	4	196.13	171.036	87.21	14556
	5	222.96	188.488	84.54	17756
	6	272.82	215.022	78.81	20084
	7	224.52	188.400	83.91	17300
	8	90.78	75.684	83.37	10412

Table 7-2: ADVISER timings for 2-cluster Cm* case.

were tried out on this example to pass beyond the requirement which enables the peak number of successful states i.e. $\psi(3,p) \wedge \psi(6,M)$. The results are shown in Table 7-2. The explosive growth in the number of polynomial terms to be processed by the merging algorithms is obvious, as is the large percentage of the total run time taken up by this merging. One encouraging note is that even in the most complex PMS structures the author input into ADVISER the problem analysis phase, before the CRPTree was finally collapsed, rarely took more than six to ten seconds of processor time. Thus if more efficient intermediate representations and algorithms could be found the overall run time even on complex problems could probably be reduced significantly.

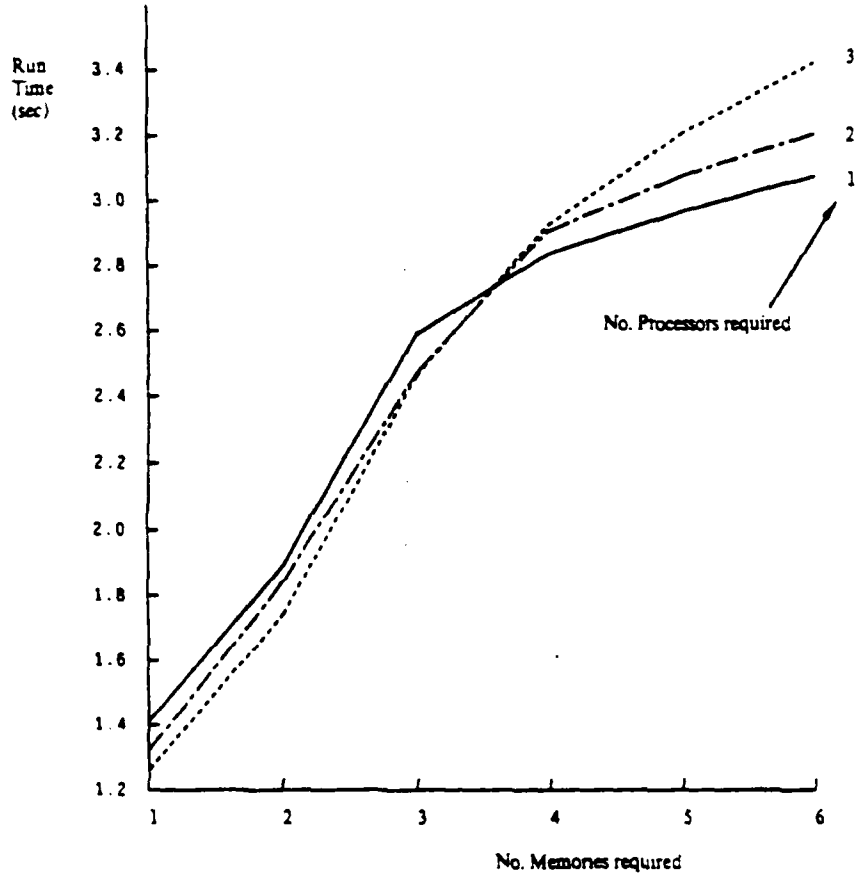


Figure 7-21: Graph of ADVISER runtimes in Table 7-1.

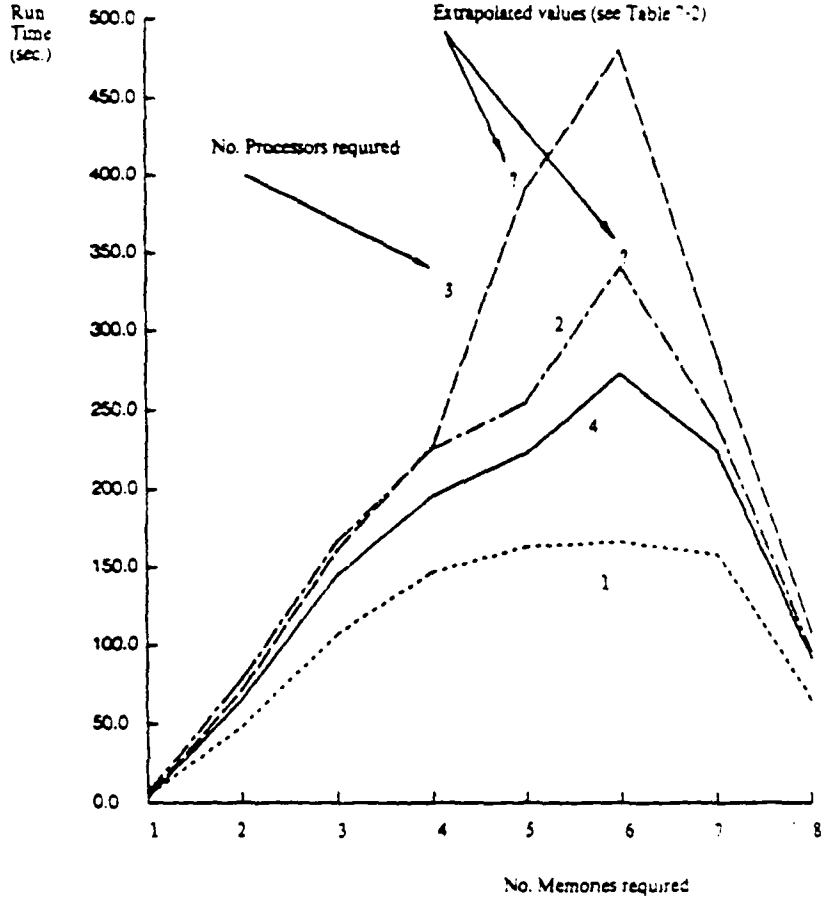


Figure 7-22: Graph of ADVISER runtimes in Table 7-2.

In the cases where question marks appear in the table the program did not complete its run due to exhaustion of memory space. This is not a serious problem necessarily inherent in the size or nature of the example but reflects inefficiency in the use and garbage collection of allocated space within ADVISER which could be improved by more careful programming[39]. In these cases the run-time figure is that which was recorded at the time of the error due to space exhaustion.

Figure 7-21 plots the values of Table 7-1 and Figure 7-22 plots the values of Table 7-2. There is a peak in the graphs of Figure 7-22 at the requirement of six memories out of a total of 12. This corresponds to the maximum number of ways of choosing some size set of memories from the 12 available. Also, the graph with the highest peak corresponds to the requirement of three processors out of six; again half the number available. A peak is not similarly visible in Figure 7-21. Since some unquantified part of the timing overheads is included in the total run time figure the absence of the peak may be due to the timing overheads being of magnitude comparable to the run time in the case of the 1-cluster Cm* case. In the 2-cluster Cm* case the run times are evidently much greater than the timing overheads and the peak is displayed. However, the timing overheads may be expected to increase with the number of CRP terms being processed and this is borne out by skewing to the right of the area under the curves in Figure 7-22.

Most of the problems chosen for the tests in Section 7.3 however were not as stressful on ADVISER. Table 7-3 shows the ADVISER run times for the examples used in the tests. For the Tandem example the table shows an additional time inside parentheses. This refers to the separate problem provided to ADVISER to compute the reliability of a "computer" (see Section 7.3.3). Although the second time is shown for both Tandem tests, the separate calculation for the "computer" reliability was carried out only once and the results used in both Tandem tests. The outstanding times in Table 7-3 are the ones for the case of C.mmp (distributed switch) and the Global Bus. The C.mmp timing for the case of the distributed switch is high since the Kernel in that case reflects the structure of the switch. This results in the generation more Kernel path reliabilities and correspondingly more PMERGE and SMERGE operations both during the computation of the path reliabilities as well as because of the overall number of CRP operations necessitated. In the case of the Global Bus reliability the example stresses the same weak point as shown above in the 2-cluster Cm* example

[39] It is possible a subtle programming error, which causes the loss of pointers to deallocated space, may be responsible since ADVISER has handled larger examples from the standpoint of lengths and number of CRPs involved

Example	Requirement	Run Time (sec.)
DEC1.PMS	1 P, 2 M	1.14
DEC2.PMS	1 P, 1 M	0.36
DEC3.PMS	1 P, 2 M	3.14
Cm*	5 P, 10 M	12.87
Cm*	1 P, 2 M	2.08
C.mmp (Lumped)	2 P, 2 M, 1 K.io, 1 K.clock	4.67
C.mmp (Distributed)	2 P, 2 M, 1 K.io, 1 K.clock	170.51
Tandem	2 C, 2 IOL	1.84 (+18.71)
Tandem	1 C, 1 IOL	5.90 (+18.71)
Global Bus	2 P, 8 M	4015.69
Pluribus	2 P, 2 ML, 2 MX, 1 CLK, 1 PID, 1 IOL	3.52

Table 7-3: ADVISER timings for architectures of Section 7.3.

(Table 7-2). In addition the addressed Global Bus structure is not regular thus resulting in fewer opportunities to simplify CRPs. Hence CRPs are in general longer in this example and since the PMERGE and SMERGE algorithms are sensitive to CRP lengths the problem is compounded.

7.5 Application to classical Network Reliability problems

In certain kinds of classical network reliability problems it is possible to use ADVISER for computing the solution. By the phrase "classical network reliability problems" we refer here to the class of problems discussed in papers such as [Wilkov 72]. The network is typically viewed in such problems as a set of homogeneous[40] processing elements or nodes connected by a set of homogeneous non-directed arcs for transferring data. Either nodes or arcs, or both, may be prone to failure with some probability. Traditional reliability measures for such networks include the probability that two particular nodes are always able to communicate, the probability that enough arcs are functional to preserve a spanning tree of the network, and so on. We note in passing here, however, that ADVISER is not constrained to the traditional assumption of homogeneity of arcs and nodes.

In order to convert such problems to the ADVISER model the treatment differs according to whether nodes or arcs, or both, are failure-prone. If only nodes are prone to failure while arcs

[40] i.e. having identical probabilities of failure or success.

are perfect then this is precisely the ADVISER model and no change is necessary to the interconnection graph of the network. In the case that nodes are perfect while arcs are failure-prone, or the case that both nodes and arcs may fail, it is necessary to convert the interconnection graph into a probabilistically equivalent network by the following transformation. If G(V,E) is the interconnection graph and the edge $(v_1,v_2) \in E$ where $v_1,v_2 \in V$ then replace the edge (v_1,v_2) by a new node v_a and two new edges (v_1,v_a) and (v_a,v_2) where v_a embodies the lumped reliability behavior of the original edge and the two new arcs are perfect. In this manner at most n(n-1)/2 new failure-prone vertices are introduced (where n is the cardinality of the original vertex set V). The new graph obtained in this fashion is probabilistically equivalent to the original graph and is composed of perfect arcs, the original nodes which continue to be perfect, and some new nodes which are failure-prone.

The kinds of requirements which ADVISER is capable of handling in such problems are any that may reasonably be transformed into the boolean expression method of specifying the reliability requirements. For instance, if it is desired to compute the probability that two particular nodes, say v_x and v_y, will be functional and able to communicate, then it is necessary to allot them distinct type names (in the ADVISER sense). Thus make v_x the only node of type TYPEX, say, and v_y the only node of TYPEY, say, in the graph. Then, to compute the desired probability, the necessary requirement to be applied to the transformed graph will be

$$\psi(1,\text{TYPEX}) \wedge \psi(1,\text{TYPEY}).$$

ADVISER will then compute the probability function using the Communication Axiom. It is possible in the case of tree-connected PMS structures that ADVISER may compute a pessimistic reliability. This case was referred to in Section 5.3.

In order to illustrate the contention of this section we choose the ARPANET example described in [Hansler 74], Page 107. The network is shown in Figure 7-23(a) where the objective is to compute the probability that vertices 1 and 8 will always be able to communicate. The vertices are homogeneous and perfect whereas the arcs are homogeneous and failure-prone. [Hansler 74] gives the following expression as the solution:

$$P_f[1,8] = 4q^2 + 6q^3 - 16q^4 - 32q^5 + 115q^6 - 134q^7 + 79q^8 - 24q^9 + 3q^{10} \qquad (7.2)$$

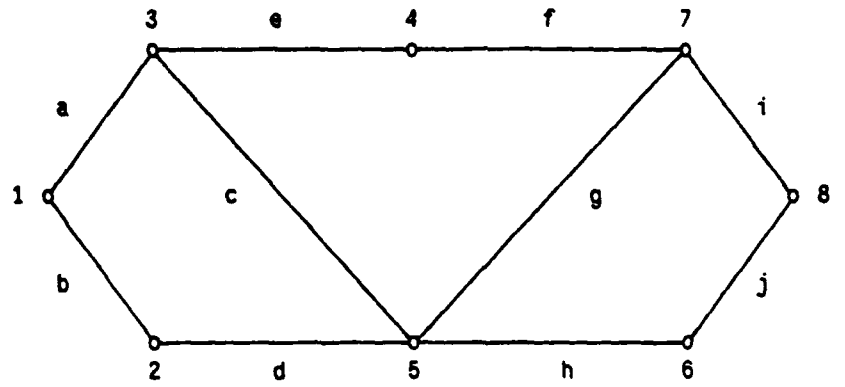
where $P_f[s,t]$ is the probability of <u>failure</u> of communication between nodes s and t and q is the <u>failure</u> probability of an arc. [Hansler 74] also defines $P_c[s,t]$ to be the probability of successful communication between s and t. Then $P_f[s,t] = 1 - P_c[s,t]$.

By the transformation outlined above we obtain Figure 7-23(b). Note that the probabilities specified to ADVISER for the arcs and nodes will be success probabilities and that the perfect vertices will be given success probabilities of 1.0. Listing 9 shows a MACSYMA session with the ADVISER solution (i.e. $P_c[1,8]$) to the problem. In line (D6) the reliability (N) of the perfect vertices in Figure 7-23(b) is set to unity. In line (D7) the success probabilities of the failure-prone vertices have been replaced by failure probabilities in order to convert the expression to the form in [Hansler 74]. It is seen that the result of the MACSYMA manipulations in line (D8) is the complement of the expression in Equation (7.2) above.

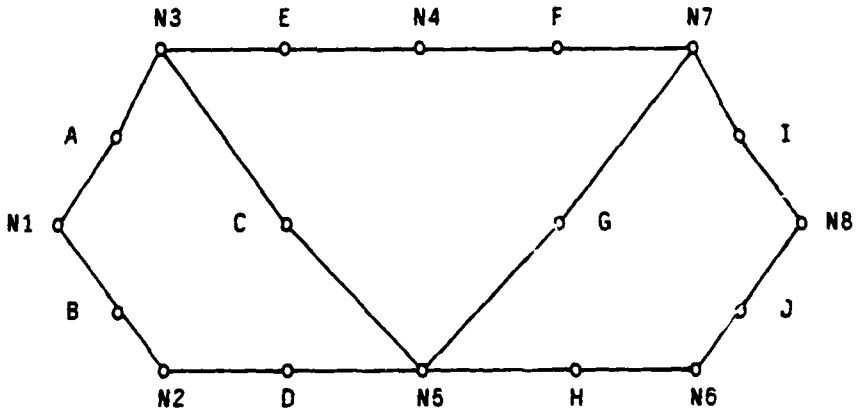
Listing 9, Arpanet Example

This listing shows the MACSYMA manipulations on the ADVISER solution to the Arpanet example of [Hansler 74] showing that it is the same solution as obtained by the latter.

```
*:a

This is MACSYMA 293

FIX293 6 DSK MACSYM being loaded
Loading done

(C1) batch(kini.eqadv);

(C2) /*
-----------------------------------------------------------------------
MACSYMA Module for Reliability Function manipulation
   produced by ADVISER on Monday, 12 Jan 81 at 12:39:21 for [4,1367]
-----------------------------------------------------------------------
Task Title: ARPANT.PMS -- A small modified Arpanet problem

Requirements on the Structure were:

(1-OF-N AND 1-OF-N)

Component-Type definitions for this task:

INDEX  TYPENAME    PRINTNAME   REL.FN.   PARAMS
-----  --------    ---------   -------   ------
    0  NODE        N           Const.    Relia.=1.00000000
    1  S           N           Const.    Relia.=1.00000000
    2  T           N           Const.    Relia.=1.00000000
    3  ARC         P           Expon.    Lambda=10.00000000

PMS Structure Definitions for this  task:
```



(a)



(b)

Figure 7-23: (a) Example network from [Hansler 74]
(b) Translation into ADVISER framework

INDEX	NAME	TYPE	NNEIG	NEIGHBORS
0	N1	S	2	(A, B)
1	N2	NODE	2	(B, D)
2	N3	NODE	3	(A, C, E)
3	N4	NODE	2	(E, F)
4	N5	NODE	4	(C, D, G, H)
5	N6	NODE	2	(H, J)
6	N7	NODE	3	(F, G, I)
7	N8	T	2	(I, J)
8	A	ARC	2	(N1, N3)
9	B	ARC	2	(N1, N2)
10	C	ARC	2	(N3, N5)
11	D	ARC	2	(N2, N5)
12	E	ARC	2	(N3, N4)
13	F	ARC	2	(N4, N7)
14	G	ARC	2	(N5, N7)
15	H	ARC	2	(N5, N6)
16	I	ARC	2	(N7, N8)
17	J	ARC	2	(N6, N8)

```
------------------------------------------------------------------------
*/

System%Reliability: 0;
(D2)                                0

(C3) System%Reliability:
       5 * N↑3 * N * N * P↑4  -  5 * N↑4 * N * N * P↑6  +  2 * N↑5 *
       N * N * P↑6  -  6 * N↑5 * N * N * P↑7  +  5 * N↑5 * N * N *
       P↑8  -  3 * N↑6 * N * N * P↑8  +  6 * N↑6 * N * N * P↑9  -  3
        * N↑6 * N * N * P↑10
       :

          8  10      8  9      8  8      7  8      7  7      7  6      6  6
(D3) - 3 N  P   + 6 N  P  - 3 N  P  + 5 N  P  - 6 N  P  + 2 N  P  - 5 N  P

                                                                    5  4
                                                                + 5 N  P

(C4) /*End of System Reliability computation*/

FACTOR(%);

        5  4     3  6      3  5      3  4      2  4      2  3      2  2
(D4) - N  P  (3 N  P  - 6 N  P  + 3 N  P  - 5 N  P  + 6 N  P  - 2 N  P

                                                                    2
                                                                + 5 N P  - 5)

(D5)                                BATCH DONE

(C6) D4,N=1;
                      4     6     5     4     3     2
(D6)                - P  (3 P  - 6 P  - 2 P  + 6 P  + 3 P  - 5)

(C7) %,P=(1-Q);
                         6          5          4          3          2
(D7) - (3 (1 - Q)  - 6 (1 - Q)  - 2 (1 - Q)  + 6 (1 - Q)  + 3 (1 - Q)  - 5)

                                                                    4
                                                                (1 - Q)
```

```
(C8) ratexpand(%);
          10       9       8       7       6       5       4       3       2
(D8) - 3 Q   + 24 Q  - 79 Q  + 134 Q  - 115 Q  + 32 Q  + 16 Q  - 6 Q  - 4 Q
                                                                          + 1

(C9) quit();

:KILL
•
```

7.6 Summary and Conclusions

This chapter has described some of the results of testing ADVISER for correctness of its output. In all of the cases so far studied, where ADVISER's deficiencies did not come into play, it has been possible to show that the chance of ADVISER's output being correct is high. In those cases where comparison to manual computation was feasible the program output was indeed verified to be correct. Further testing using more complex and stressful examples will be required before strong confidence in the program is justified. However, the prognosis for the program's useability appears very good.

The major hindering factor at this stage of ADVISER's development is the inherent inefficiency of its low level PMERGE and SMERGE algorithms for dealing with the intermediate representation. These inefficiencies were partly to be expected with the use of canonical forms of polynomials, however, what was quite unexpected was the large percentage of the typical program run time consumed by these algorithms. The encouraging thought is that the problem analysis time, preceding the collapsing of the CRPTree in the final phase of the computation, appears to take typically around 15 percent of the solution time. Improvement in the intermediate representations and algorithms can therefore be expected to bring about signifcant improvements in the program's performance. Minimal forms instead of canonical forms of expressions must be used. Improvements must be obtained to algorithms such as those described in [Bennetts 75] and [Satyanarayana 78] so that they may be incorporated into ADVISER and used at all stages of the computation rather than in a final pass over a constructed intermediate representation.

A solution to the deficiency in the bounded-cluster side constraint specifiability is fairly straightforward and requires mostly an implementation effort. However, the deficiency in the TREEREL algorithm is more severe and will need more research to arrive at a satisfactory solution. Despite these deficiencies the program can be useful for analyzing the reliability of many types of PMS structures.

Chapter 8
Summary, Conclusions, and Future Research

The work contained in this thesis has been an attempt to introduce a new and higher level approach to the reliability modelling of computer structures at the *Processor-Memory-Switch (PMS)* level of design. Traditionally, reliability calculation programs have addressed the problem largely at the level of analysis of fault-trees and reliability graphs (for example see [Misra 70a], [Barlow 75b], [Chelson 71], [Fleming 71], [Bennetts 75]). The fault-tree or reliability graph is assumed to have been derived by hand from the program-user's knowledge of the system being analyzed. In another branch of this endeavor, systems are viewed as cascades of GMR (General Modular Redundant) subsystems, for example see [Mathur 75a], [Ng 80]. In these cases the user of the reliability calculation tool is required to first appropriately segment the system under consideration into such subsystems. Often this may not be possible. Much work has been done toward the calculation of what is commonly termed "network reliability" in the literature in one of two senses. In one sense this term nas been used to refer to computing of system reliability as encoded by reliability graphs. In the other sense the term refers to the calculation of reliability of such loosely coupled systems as geographically distributed computer communication networks (see [Wilkov 72]). The underlying model in these latter instances has generally been a graph with homogeneous vertices and arcs, where the vertices or the arcs, or both, are prone to failure. The reliability measures of interest in these cases have tended to focus, for instance, on the probability that certain key vertices in the network are able to communicate at all times. In the case of PMS systems the vertices of the interconnection graph represent the non-homogeneous components in the system and therefore must be labelled to reflect this non-homogeneity. Furthermore, the criteria for system functionality are more complex and require typically that a certain minimum sized assortment of types of components be functional and capable of intercommunication.

The main goal of the work has been to develop techniques which enable much more of the PMS system reliability calculation process to be automated than has heretofore been customary. The concomitant desirable result of achieving such a goal is the reduction of the

possibility of error which is ever-present in tedious hand calculation. Furthermore, relatively unsophisticated modellers of PMS system reliability have access to a powerful tool which relieves them of the burden of attending to many of the complexities of such modelling.

This dissertation describes significant progress toward such a goal. The ADVISER program was constructed to provide a reliability calculation aid at the PMS level. The program accepts as one of its inputs the interconnection structure of the PMS system in the form of a graph in which the vertices are labelled with the generic type of the components they individually represent. Another input is a set of requirements or criteria for system functionality in the form of a modified Boolean expression. Further *ad hoc* side-constraints may also be provided by the user. The output of the program is a text file which contains a program to compute the reliability function of the described PMS system under the given requirements and constraints.

The next section will recapitulate the material of the chapters of this thesis taken in sequence and the final section will restate the problems which remained unsolved at the conclusion of this investigation and propose areas for future research in the field.

8.1 Recapitulation

Chapter 2 reviewed earlier work on algorithms for reliability calculation and various efforts towards building software design tools which calculate the reliability of various kinds of systems. It was noted that while many such reliability calculation aids had been constructed virtually every one of them has required the user of the program to do a partial analysis of the system under consideration. Typically, then, some intermediate representation is generated by the user which encodes the result of his system analysis. Depending on the type of system reliability analysis desired this intermediate representation is usually either a fault or event tree or a reliability graph. Having derived the intermediate representation the user then proceeds to feed it to one of several commonly available reliability calculation aids which use the intermediate representation to compute numerically, or symbolically, the system reliability.

This perspective lead to the posing of the question: *Is it feasible to construct a design tool which will compute the symbolic reliability of PMS level systems directly from their interconnection graphs and a statement of the criteria by which they are judged functional?* This investigation undertook to answer the question. The rest of Chapter 2 provided a broad overview of the functioning of the ADVISER program which was the result of the investigation.

The program operates on what are termed Canonical Reliability Polynomials (CRPs). These are the equivalent of an intermediate representation mentioned above. It incrementally generates CRPs as Partial Results for each class of functional system states which arises as a result of its case analysis of the PMS system. The structure of a CRP bears a close resemblance to the structure of a Boolean expression in disjunctive normal form except that the literals in the CRP are the symbolic probabilities of success (reliabilities) of unique components in the structure. CRPs can be "merged" in two ways which respectively represent the computation of the probabilities of intersections (SMERGE) and unions (PMERGE) of events. The primitive events under consideration are the successes of components in the system and system success is a compound event. Chapter 3 described the list data structure used to represent CRPs and algorithms for the SMERGE and PMERGE operations. The algorithms are simple and robust in the face of overspecification. In other words they will correctly merge two CRPs to form the appropriate CRP representing the intersection or union of the events represented by the CRPs which were merged, regardless of the exact set of events or antecedents of the merged CRPs. They are thus ideal for use in a program such as ADVISER wherein CRPs are generated in several different phases during a program run and no record is kept of how any particular CRP was generated. The drawback with these algorithms is their time complexity which is $O(n^2)$ if each of the list data structures representing the two CRPs to be merged is of length $O(n)$. Furthermore, the list resulting from a merge has a length of $O(n^2)$ thus making successive merge operations more and more expensive. This is the major drawback of the ADVISER program as it is currently constituted and some palliative measures are reported in Chapters 3 and 6. The SMERGE and PMERGE operations are used throughout the rest of the program to appropriately combine partial results in order to finally produce the system reliability function.

Chapter 4 described symmetry detection algorithms, based on the work of Gaschnig, [Gaschnig 77], which are used to detect symmetric subparts of the PMS interconnection structure. The motivation is to employ any existing symmetry to advantage by computing results for only one of a set of several symmetric subparts and applying those results identically to the rest of the members in the set. The result of processing the interconnection graph of the PMS structure through the symmetry detection algorithms is a Typed Neighbors Class Graph (TNCG) whose vertices are the symmetry classes induced by the Typed Neighbors Class Equivalence Relation (TNCER). An examination of the TNCG reveals the nature and number of symmetric substructures within the PMS system.

It was postulated that a divide-and-conquer approach to the system reliability calculation would be fruitful if the program had a repertoire of special reliability calculation techniques for

various kinds of substructures within a PMS system. Then the segmenting of the PMS interconnection graph to provide the subproblems for the divide-and-conquer paradigm could be done on the basis of substructures for which special techniques were known. Quite by chance one of the most frequently occurring substructures in PMS systems is the tree interconnection structure. Chapter 5 describes the GROWTREES algorithm whereby Pendant Tree Subgraphs (PTSs) are recognized in the TNCG, thus providing a knowledge of symmetric PTSs in the original PMS interconnection graph. The TREEREL algorithm is also described and embodies the special techniques mentioned above for the case of tree interconnection structures. Using the SMERGE and PMERGE algorithms of Chapter 3 it computes the symbolic reliability for a PTS given the interconnection structure of the PTS and criteria for its functionality. In this respect the TREEREL algorithm is a microcosm of the ADVISER program.

Chapter 6 describes the OVERLORD routine in the ADVISER program which orchestrates the functions of the various algorithms described in earlier chapters. After the PMS structure has been input along with the requirements the OVERLORD routine takes control. It invokes the SYMMDET algorithm on the PMS interconnection graph and then proceeds to discover symmetric PTSs in the graph by calling the GROWTREES algorithm. It then invokes the TREEREL algorithm a sufficient number of times on the discovered PTSs in order to precompute all partial results (CRPs) for PTSs which may be expected to be used during the remainder of the program run. Finally it proceeds to fragment the input requirement into all possible subcases which the system can satisfy and, thus, be functional. Cases of interest are those in which the functional components in the system satisfy the Communication Axiom and other side-constraints which were specified by the user of the program. Each of these cases actually represents a class of functional system states since the enumeration is done over functional substructures of the PMS system rather than over individual components. For each case it generates a partial result or CRP and finally merges these partial results to form the CRP which represents the system reliability. During the last phase of the program run this final CRP is algebraically simplified using what is known about the generic class of components to which each component in the PMS system belongs; all components in the same generic class are deemed to have identical reliability functions. The simplified reliability function is then printed out as a file containing the text of a FORTRAN or SAIL procedure which numerically computes the reliability function just derived.

Chapter 7 presents some experimental results obtained by using the ADVISER program with some typical PMS structures. In an effort to engender confidence in the correctness of ADVISER output the output was checked in two ways. The first check involved comparing the

output for a PMS structure, which could be analyzed easily by hand, with the manually computed reliability function. In the second form of check numerical values of system reliability were obtained using the reliability function output by ADVISER. Programs written by independent researchers to calculate the reliability specifically for those systems were used to obtain a second set of values against which ADVISER values were compared. Both forms of test were successful in showing that ADVISER output was correct with high probability and that the program could be useful in analyzing a variety of architectures. Experiments in analyzing the performance of ADVISER were not as satisfying, perhaps inevitably so, given that the ADVISER software is of necessity not of production quality. Large percentages of the compute time for a problem are spent in the low level algorithms which operate on the intermediate representation. There is clearly much room for improvement in these algorithms and the indications from the experiments are that modest improvements in them would result in significant improvements in program performance.

8.2 Future Research

There are two general classes of topics for future research in the area addressed by this dissertation. The first of these is the set of problems which were encountered while constructing the ADVISER program as a test bed for the ideas developed here and for which no solutions have yet been found. The other class of topics is generated by the systematic elimination of the fundamental underlying assumptions which were made in order to circumscribe this work. We shall discuss both these classes of problems in the following.

8.2.1 Unsolved problems in the present framework

8.2.1.1 Intermediate Representation

The Canonical Reliability Polynomial (CRP) as an intermediate representation in ADVISER provided some attractive benefits from the point of view of the design and construction of the program. The algorithms to process CRPs are simple and robust and will work to correctly merge any arbitrary set of CRPs so long as the set of primitive events is fixed and CRPs for all compound events are generated only by means of the merge algorithms themselves. Thus the software package which handles CRPs as the primitive operands could be designed completely independently of any other part of the program and be called during any phase of the program run. Ironically this very simplicity gives rise to a combinatorial explosion in the case that the input requirements expression is exceedingly complex or in the case that for any critical component type the requirements demand that about half of the available components

of that type in the structure be functional. In other words the program performs poorly when it is operating near the peak of the binomial distribution of the number of functional subsets of components in each critical component type.

The primary cause for this undesirable time complexity is that the reliability polynomials are maintained in canonical form and the distinct identities of the individual components are maintained in the symbols which express their reliabilities. The former fact implies that list lengths will be longer than for the equivalent polynomials in factored or minimal form, thus adding to the complexity. The latter fact implies that, for instance, cancellations of a pair of terms will not take place even though this would happen if generic reliabilities were to be substituted for the unique component reliability symbols in those two terms. However, it is also to be noted that the retaining in the CRPs of the unique component reliability symbols allows the easy computation of probabilities of unions and intersections of events. Factored or minimal forms of the polynomials will not be so amenable.

Of the unsolved problems at hand, therefore, perhaps the most important is to devise a suitable intermediate representation and algorithms to operate on it efficiently. Any such new method for the manipulation of symbolic reliability expressions must be of some form which does not have the above disadvantages, while it maintains the advantages of the CRP approach. These advantages are

- Robustness in the face of overspecification in the input requirements
- Inherently simple algorithms, and
- May be used at any phase of the computation rather than once at the end.

The performance measurements of Chapter 7 indicate that if such a method can be found it would drastically improve the useability of the ADVISER program. Of significant interest for modification and adaptation to the ADVISER framework are the kinds of algorithms described in [Satyanarayana 78], [Bennetts 75], [Aggarwal 78] and [Lin 76].

8.2.1.2 The CRPTree

During the generation of the system CRP the various other CRPs, which were precomputed and stored in hash tables by the program, are used several times in the merging process. As we saw in Chapter 3, however, the amount of compute time consumed in the CRP package is $O(n^2)$ where n is the length of the CRP term list. The lengths of the lists also grow each time a merge operation is performed. Hence the more merge operations are performed, the slower the program runs.

The CRPTree was used as a device to ensure that the merging of any given CRP would be performed as late and as few times as possible in the process. This was achieved by delaying the merging until after the CRPTree was built. Then for each node of the CRPTree, the CRP which labels it would be merged only once with the CRP which results from the "collapsing" of the subtree beneath that node. However, as the "collapsing" process reaches the upper levels of the tree, the lists have typically already grown to an undesirably large size. There is thus an incentive to process the CRPTree more efficiently than at present. It is possible in the case of highly symmetric PMS structures such as Cm* that the CRPTree will itself contain symmetries. These must be exploited. For instance if two subtrees of the same CRPTree node are symmetric in the sense that they will produce identical CRPs after collapsing, then only one of them need be collapsed since the other will be superfluous due to the idempotency of the SMERGE and PMERGE operations. Even if two symmetric subtrees in the CRPTree do not share the same immediate ancestor, a copy of the CRP resulting from one may be used for the other. Other efficiencies may also be possible. Methods for efficiently collapsing the CRPTree would add further to the efficiency and useability of ADVISER.

8.2.1.3 Side Constraints

The set of side-constraints suggested in Chapter 6 is *ad hoc* although it appears to be sufficient for a wide class of examples. An open area of investigation is the determination of whether this set of side-constraints can be extended or made more sophisticated or unified in a theory of constraints.

8.2.1.4 Enhancement of TREEREL algorithm

The TREEREL algorithm described in Chapter 5 expects a Pendant Tree Subgraph under the implicit assumption that all components within the PTS and the Kernel communicate through the root vertex of the tree. If the given input PMS structure *is itself a tree* to begin with, then a possibly pessimistic reliability estimate results since, in order for two components to communicate under the model, the communications are assumed to take place through the root of the PTS even though the two components are in the same subtree of the PTS and can communicate via a more direct path through the root vertex of that *subtree*. As was seen in the Cm* example of Chapter 7 this deficiency comes into play also in cases where the entire requirements expression can be satisfied by some subtree of a PTS segment of the overall PMS structure. Some variant of the current TREEREL algorithm needs to be devised to eliminate this deficiency. This is not viewed as an exceedingly difficult task.

8.2.1.5 Further exploitation of symmetry

At present the ADVISER program makes use of inherent symmetry in the PMS structure only in order to discover symmetric PTSs so that redundant computations on such PTSs can be avoided by doing them for only one of each symmetric set. However, as in the case of the Cm* structure (Section 7.3.1), there may be considerable symmetry *within* a PTS which could be exploited and is currently not. In effect for simple cases of symmetry such as in the Cm* architecture closed form solutions are readily available. The question is one of efficient representation and storage of such closed forms. If such closed forms are to be used, then related questions arise regarding the ease of computation of the probabilities of intersections and unions of events represented by them. These impinge on the design of the intermediate representation and efficient algorithms to manipulate it. Also to be answered is the question whether ADVISER should be outfitted with an *ad hoc* collection of closed form solutions for specific instances of symmetric structures; to be used whenever any such structure is recognized. It would perhaps be preferable instead to devise and incorporate an algorithm which would recognize cases for which a closed form solution could be constructed and do the construction from first principles.

In a similar fashion inherent symmetry in the Kernel could be taken advantage of in the path reliability computation. However, the advantage in this case may not be as pronounced since the Kernel would tend to be small in most cases.

8.2.2 Relaxing of Underlying Assumptions

The relaxing of the assumptions, which underly the present work and which were stated in Chapter 2, would be the next logical step in the continuation of the research effort in this field. Some areas of investigation, spawned by such a loosening of assumptions, are mentioned in the following.

8.2.2.1 Directed Graphs

The current model of PMS systems which is built into ADVISER is that of an interconnection graph which is non-directed and having labelled vertices. One may seek to relax this restriction and allow directed graphs to be introduced. Doing this does not have as great an impact on the model as might be imagined. For instance, the path-finding algorithm PATHREL, which computes the Kernel CRP, would change only very slightly. Potential paths to the goal vertex would be sought only along arcs leading away from the current vertex. The function of the Internal Port Connection Matrix (IPCM) would remain exactly the same as it is

at present[41]. The symmetry algorithms of Chapter 4 would generalize to the case of directed graphs; the algorithm described in [Gaschnig 77] was in fact originally derived for the general case.

One model which would change upon the introduction of directed graphs would be that of Pendant Tree Subgraphs (PTSs). It would still be possible to generate these PTSs in similar fashion if the direction of an arc connecting two neighboring vertices is disregarded. The current model assumes, however, that all communication amongst components in the PTS occurs through the root vertex of the PTS. This may not necessarily be the case in directed graphs. One solution might be to let the GROWTREES and TREEREL algorithms work exactly as they do now and apply them only to those PTSs which are such that all pairs of vertices in them are joined by dual directed arcs, one in each direction (this is essentially equivalent to the directed case). However, a possible implication may be that at most times the program will be incapable of generating any segments except a Kernel which contains essentially the whole graph itself. This defeats the purpose of the special case solvers. The better alternative then is to devise tree algorithms which construct symbolic reliability functions for trees which are directed graphs.

8.2.2.2 Statistically dependent component failures

One of the underlying assumptions of the present framework was that the failure behaviors in the PMS system components are statistically mutually independent. The relaxing of this assumption, to include cases in which statistically dependent failure behavior is possible. implies that the joint failure probabilities of pairs of components would have to be taken into account. A possible initial solution to this problem is suggested by the nature of CRPs. Recall that the juxtaposition of factors in a CRP term represents the SMERGE of the corresponding probabilities and not just a multiplication; it is treated throughout this thesis as eventually being a multiplication only because of the basic assumption of s-independent component failure behavior. The SMERGE operation computes the probabilities of the intersection of events and one can envision constructing an SMERGE algorithm on CRPs for the situation where the joint failure probabilities of pairs of system components are non-zero. In this situation one may find factors in a CRP term which represent such dependent components. If the symbolic joint failure probabilities of these components are known then the intersection probability (i.e. the probability of the simultaneous functioning of these components) may be computed using the laws of probability.

[41] The current version of ADVISER does in fact allow non-symmetric IPCMs to be specified for components within the Kernel and the path reliability algorithms do take this into account but directed graphs are not handled in their generality.

Care needs to be exercised in dealing with CRPs in such a situation. For instance, many of the procedures described in earlier chapters, such as the simplification of the system CRP, assume implicitly that any CRPs derived for two disjoint sections of the PMS graph, are independent. These would have to be modified to accommodate the general case. The problem of handling dependent failure behavior is still open, however, and points out one more attribute which must be possessed by any intermediate representation formalism which is sought to replace the CRP.

8.2.2.3 Coverage factors

A major simplifying assumption of the current model has been that coverage of hard failures of system components is perfect. In other words it has been assumed that the conditional probability that the system will recover from component failures without loss of information is unity. This is clearly optimistic. It has been shown by other investigators [Bouricius 69] that overall system reliability is very sensitive to coverage. A more accurate modelling of PMS system reliability would require that imperfect coverage be accounted for.

The CRP intermediate representation uses only the success probabilities of the system components. Therefore, the inclusion of coverage factors into the CRPs being computed is not straightforward since coverage factors apply upon the *failure of system components*. The formulae described in Chapter 3 which form the bases for the current CRP algorithms may be modified to include coverage factors as follows:

$$Pr\{ev_A \cap ev_B\} = R_A \otimes R_B \qquad \text{(SMERGE)}$$

$$Pr\{ev_A \cup ev_B\} = R_A \otimes C_B(1-R_B) + R_A \otimes R_B + R_B \otimes C_A(1-R_A) \qquad \text{(PMERGE)}$$

where C_A and C_B are the coverage factors for the events ~A and ~B respectively. The second of the two equations above reflects the fact that the union of the events A and B is composed of the three mutually exclusive events (A $\wedge$ ~B), (A $\wedge$ B) and (~A $\wedge$ B) respectively. The inclusion of the coverage factors causes the PMERGE to be explicitly a reliability computation rather than just the computation of the probability of an intersection of two events as in Chapter 3. In the above equations, however, it is also apparent that coverage factors will need to be known for all possible compound events in the event space. Unfortunately, in general systems the estimation of coverage factors is at best a difficult proposition since the factors are so dependent on things other than the hardware of which the system is composed. Coverage factors are more likely to be known in the case of failures of entire subsystems

rather than at the individual component level. In addition the current data structure used for CRPs will not suffice since it is oriented strictly toward representing success probabilities. The devising of CRP algorithms, or algorithms on other intermediate representations, which incorporate coverage factors in the computation of system reliability, is a research area of prime importance.

8.2.2.4 Multi-state models of component reliability

The current model, on which the ADVISER program is based, assumes that any system component may be in one of only two possible states, i.e. failed or working. No allowance is made for components whose failure behavior is characterized by transit through a sequence of states in which the component is successively more degraded. For instance it may be possible that partial failure of a component manifests itself as degraded performance, which in consequence similarly degrades the performance of the PMS network. If the PMS system performance, according to some metric, is required to be greater than some minimum for the system to be functional (see 8.2.3.1 below) then such partial failures of components are of importance in determining system reliability. The adoption of a multi-state model of component failure behavior, however, invalidates the simple boolean requirement specification method along with the CRP representation which both depend on the binary state model.

8.2.3 Other research issues

8.2.3.1 Incorporating performance into system reliability

As was mentioned above, one may consider the reliability of a PMS system to be a function not only of the hard failure reliability of its individual components but also of their capability to process information at rates above some decreed minimum necessary for system functionality. Recent work has been reported in the construction of such compound models ([Beaudry 78], [Castillo 80]). Typically in such a model one would consider components which degrade in their performance in several steps before failing completely. When the reliability measure of a PMS system is to include performance issues, the flow rates between components in the structure assume importance. In such situations, in addition to requiring that a certain assortment of critical components be at least minimally functional and able to communicate (as in the current ADVISER program), one would require not only that the individual critical components support a certain minimum information processing rate but that all the functional paths used for communication between them also sustain certain minimum

data rates at all times. Thus the general max-flow-min-cut algorithms (see [Ford 62]) would become applicable in the reliability analysis and, for instance, the path-finding strategy in the Kernel (see Section 6.6) would have to be enhanced to use them. Such concerns would affect the method of requirement specification in the ADVISER framework. It will no longer be sufficient to require a minimum number of components of each critical type to be minimally functional. Either by including a separate specification, or by modifying the current specification method, a further requirement for minimum sustained information flow rates per critical component type perhaps, would need to be specified.

In addition to the above, once information on flow rates has been included in the PMS structure which is described to the program it will also become possible to check for inconsistencies in the way the system is structured in that the specified minimum required information flow rates may not be supportable at all. Such "go-nogo" tests would be useful to the designer trying to construct a PMS system from a database of components to specify given design constraints on performance and reliability.

8.2.3.2 Other special solvers

Given the acceptability of the ADVISER method of generating reliability functions for PMS structures, further research needs to be carried out into other kinds of substructures which might appear with some regularity in extant systems. The nature and frequency of such substructures might change with time and with the evolution of design principles and systematic design methodology for PMS structures. Further algorithms on graphs need to be developed to discover the existence of such substructures in the original PMS interconnection graph. The greater the number of special-case structures for which solvers are available, the smaller it is hoped the typical Kernel would become. Current practice in ADVISER is to assign unique single-symbol aliases to partial result CRPs emanating from the special solvers. Adherence to this along with the simpler path-CRPs from a smaller Kernel could act to cause CRPs to be shorter thus decreasing the time consumed in the CRP algorithms.

8.2.3.3 Indefinite requirement specifications

The current specification of the requirements for system reliability use a constant requirement integer. Thus, for instance, a typical atomic requirement of the form ψ(10,Processor) requires that at least 10 Processors be functional. However, the truly general case would imply the requirement of an indefinite number of Processors. Thus, for example, ψ(q, Processors), where at least q Processors are required for some q. Naturally the lower

bound on q is zero and the upper bound is the total number of Processors contained in the PMS system under study. If the system reliability function were produced in terms of such indefinite requirements, then parameterized studies of PMS system reliability would be facilitated ever further. The problems in implementing such a capability in an ADVISER-like program are difficult and would probably require a lot of the sophisticated symbol manipulations available in a program like MACSYMA, [Macsyma 77].

An additional enhancement to the requirement specification, which could probably be of much use to users of programs like ADVISER, would be the ability to refer in the requirements expression to *specific* components in the PMS structure by name. This may be done in a round about fashion in the current implementation by isolating the specific component of interest into its own component-type category and then requiring at least one component of that type to be functional. However, the ability to attach special significance to a member of some component type would cause the treatment of the components of that type to be non-homogeneous. This would probably necessitate the redesign of many of the algorithms described in earlier chapters.

8.2:3.4 Reliability models for repairable systems

In the case of repairable systems the problem of reliability calculation is faced with two random variables, namely the time to failure and the time to repair. This complicates the mathematical analysis significantly. Several other issues which strongly affect the tractability of such problems need to be addressed. For example, the sharing of repair facility amongst components causes their reliabilities to depend on each other in complex ways. Generally, Markov models are required to characterize the failure behavior of such systems and to calculate their reliability and availability. Closed form solutions may or may not be available depending on the complexity of the interdependence between components. It is not apparent at present how the ADVISER framework may be modified to solve such problems or whether the framework is adequate at all in the general instance.

It may be useful to point out, however, that ADVISER as it is currently constituted can be used to compute PMS system availabilities in a very constrained case of repairable systems. This case is one in which *each* individual system component is endowed with a dedicated repair facility which is completely independent from all the other repair facilities of other components. This assumption preserves the independence of the primitive events in the current ADVISER framework. Consequently the symbolic probabilities in the final system CRP computed by ADVISER may just as well be viewed as availabilities (transient or limiting) or as reliabilities.

8.3 Summary

This thesis has reported work towards developing a strategy for the automatic generation of symbolic reliability functions for Processor-Memory-Switch structures. A program named ADVISER embodying the resulting ideas was described and some details of its implementation were given. Algorithms which were used during the various phases of the computation in ADVISER were described and their advantages and disadvantages discussed. The presentation ended with the description of example PMS structures which were run through ADVISER and a discussion of some fruitful areas for future research.

Appendix A
A special case of inputs to PTS algorithms

Chapter 5 assumed that the PMS interconnection graph G was not a tree although that it did possibly have PTSs. A question arises as to what the response of the PTS algorithms of Chapter 5 will be if G is *itself* simply a tree graph. Under these circumstances there is no Kernel (see Chapter 2) and computing the reliability of the system implies using the TREEREL algorithm of Chapter 5 on G. However, this algorithm assumes the existence of a distinguished or root vertex of the tree upon which it operates. In the case that G is simply a tree there is no one vertex which can be termed an "interface vertex" to the Kernel (which does not exist in this case) and the tree is not rooted. Likewise, Algorithm GROW attempts to "grow" each germinal PTS towards a putative root vertex and one might question the result when GROW is applied to an unrooted tree-structured PMS system. It so happens that Algorithm GROW will work even in this case. It will, in fact, identify the entire tree graph G as a "PTS" and choose a specific vertex as its "root". The properties of this vertex are the subject of Section A.1. In the case of the TREEREL algorithm, however, the resulting computed reliability could be pessimistic and a discussion of this possibility was provided in Section 5.3.

A.1 Special case operation of Algorithm GROW

The GROW algorithm operates on G′, the NCG of G, which in this case is also a tree graph. For G′ to be a tree graph it is sufficient (although not necessary) for G to be a tree graph. Two cases can be identified here.

1. G has at least one vertex with two or more symmetric subtrees below it. This allows the symmetry detection algorithms to equivalence, or "fold" them.

2. G has no vertices with symmetric subtrees below them.

In the former case, because of the equivalencing or "folding", there will be at least one connection density in G′ which is greater than one. In the latter case, all the connection densities labelling the arcs in G′ will be identically unity since each vertex of G will occupy its own equivalence class alone due to the lack of any symmetry. We shall consider Case 2 first.

A.1.1 Connection densities all unity

As usual, the set of pendant vertices of G′, which are also pendant vertices of G, are chosen as the germinal trees. Then during each iteration of the algorithm some of the germinal trees will grow by one edge toward some vertex which the algorithm will eventually identify as the "root" of the "PTS". This vertex will be the root of the tree (G′ as it turns out) which is formed by the coalescing of the two or more germinal trees in the final iteration of the algorithm. It is clear that none of the germinal trees will have the function "MarkComplete" called on it[42] because, for any $t_r^{(i)}$ the connection density to all neighbors will be unity and there are no self loops in G′. Hence the algorithm will only complete when all the germinal trees have merged into one.

> **Theorem 8.1:** Let G(V,E) be a tree graph which is the interconnection graph of a PMS structure. Let G be such that G′(V′,E′), its TNCG, has all the connection densities labelling edges in E′ identically equal to one (i.e. G′ is isomorphic to G). Then Algorithm GROW will pick a root vertex $v_r \in V$ with the following property. If q_r is the length of the longest path of which v_r is one terminal vertex and if q is the length of the longest path in G, then
>
> $$q_r = \begin{cases} q/2, & q \text{ even} \\ \lceil q/2 \rceil, & q \text{ odd} \end{cases}$$
>
> Furthermore, if there are $m \geq 1$ longest paths in G, each of length l, and composed of vertex sets $V_1', V_2', \ldots, V_m' \subseteq V$ respectively, then $v_r \in V_1' \cap V_2' \cap \ldots \cap V_m'$.

We shall first introduce two necessary intermediate results.

> **Lemma 8.2:** The end points of any longest path in a tree graph are leaves of the tree.
>
> **Proof:** The proof is obvious.
>
> **Lemma 8.3:** If G(V,E) is a tree graph which has m>1 longest paths of length q and vertex sets $V_1', V_2', \ldots, V_m' \subseteq V$ respectively, then $V_1' \cap V_2' \cap \ldots \cap V_m' \neq \Phi$, i.e. they have at least one vertex in common.
>
> **Proof:** The proof is by contradiction. Assume that there are two longest paths p_1 and p_2 in G, each of length q and having vertex sets V_1' and V_2' respectively. Assume they have no vertices in common i.e. $V_1' \cap V_2' = \Phi$. Since G is a tree and is connected there must be a path of at least one edge, e′, between *exactly one* pair[43] of vertices (v_a, v_b), $v_a \in V_1'$, $v_b \in V_2'$. In the worst case let us assume that v_a and v_b occur exactly at the midpoints of the paths p_1 and p_2 respectively. Then the existence of e′ assures that there is a path of length $q/2 + q/2 + 1 = l + 1$ in G. Thus p_1 and p_2 are not longest paths of G.

---

[42] See pseudo-code for Procedure GROW.

[43] for otherwise G would not be a tree

Proof: of Theorem 8.1. In any iteration Algorithm GROW adds exactly one vertex v' and, therefore, exactly one edge to a germinal tree t'' which is still capable of growing. This vertex v' is added to the germinal tree if and only if the connection density to it from the current root of t'' is exactly unity. In addition, v' must be the only neighbor vertex of the root of t'' which has not already been precluded from consideration by inclusion in t'' or some other germinal tree. This ensures that a vertex v' of G which is eligible for inclusion, but not already included in some germinal tree, will be included only when the following condition is met: All germinal trees which will eventually coalesce into a germinal tree t' with root vertex v' must have grown to within exactly one edge of v'. Thus the inclusion of vertex v' will be delayed until its germinal subtree of greatest height (i.e. path length from that subtree's root vertex to its leaves) has grown to within one edge of v'. This is because germinal trees grow by at most one edge every iteration. Hence the number of iterations required to grow some germinal tree t' is equal to the length of the longest path from its root v' to its leaves.

Assume initially that there is exactly one longest path p' of length l in G. Its end vertices according to Lemma 8.2 are leaf vertices of G. Since germinal trees are started with the leaf vertices of G there will be at least two germinal trees growing toward each other from opposite ends of p' Since at most one edge is added during each iteration to either of them, these two germinal trees will finally coalesce after $q/2$ iterations if q is even. At that time the root vertex of the single remaining germinal tree will be the center of the longest path in G i.e. the longest path from the root will be $q/2$ long. If q is odd then these two germinal trees will approach each other until their root vertices are neighbors. Then, one of the two vertices will have to be chosen over the other as the final root. Hence the longest path length starting at the final root will be $\lceil q/2 \rceil$. Furthermore, by Lemma 8.3 the final root will be on all the longest paths in G since those germinal trees would have coalesced all together after $q/2$ iterations ($\lceil q/2 \rceil$ if q is odd).

It will be seen from the above, therefore, that the algorithm will "deadlock" in its choice of a root vertex just before the final iteration if the following conditions are met

- The longest path length, l, in G is odd. (Hence there can be at most two germinal trees left to coalesce in the final iteration.)

- The connection densities between the roots of the two germinal trees before the final iteration are both unity.

In this case, were the final iteration to follow, each germinal tree would include the root of the other into itself thereby causing a situation that is contradictory since the final root is indeterminate. This is resolved in the program by modifying Procedure GROW as follows. The germinal trees during any iteration are treated as an ordered tuple. Then when making the pass over the tuple which checks whether a vertex may be added to any of the trees, a further check is made. For tree $t^{(i)}$, if a suitable neighbor, v_n, of $t_r^{(i)}$ has been found which may displace $t_r^{(i)}$ as the new root, the algorithm also checks to see that v_n is not already included in some tree *of lower order in the tuple than* $t^{(i)}$. This check is suggested by the proof of Theorem 8.1 above. It will cause the "deadlock" to be broken in all cases of odd maximum path length.

A.1.2 Connection densities not all unity

In the other case mentioned above, the NCG of G when G is a tree may also contain connection densities greater than one. When this happens it is possible that some germinal tree will be prevented from growing because all of its connection densities to its neighbors may be greater than unity. In this case Theorem 8.1 will no longer apply. It is difficult to predict in the general case, which vertex of G will be chosen as the root by the algorithm since it depends entirely on the distribution of the non-unity connection densities within G′.

Appendix B
Terminology

Page numbers refer to the page of definition or first use of the terms in this list.

Atomic Requirements
These are requirements which are clauses of the form "at least N components of type X need to function". The clauses are abbreviated $\psi(N,X)$. Page 24.

Conjunctive Requirements
These are requirements which are pure conjunctions of Atomic Requirements. Page 34.

Disjunctive Requirements
These are requirements composed of conjunctions and/or disjunctions of Atomic Requirements Page 34.

RBD Reliability Block Diagram Page 51.

SPRBD Series-Parallel Reliability Block Diagram. Page 52.

SIP operator
Symbolic Intersection Probability operator, which is denoted in the thesis as "$\otimes$". It computes the intersection probability of two events given their individual symbolic probabilities of occurrence. Page 56.

CRPs Canonical Reliability Polynomials Page 56.

NORMVEC
The NORMVEC bit vector is one of two bit vectors which can be present in a term of a Canonical Reliability Polynomial. Each bit in the NORMVEC represents a unique component in the PMS structure for which the reliability function is being derived. Page 59.

AUXVEC The AUXVEC bit vector is one of two bit vectors which can be present in a term of a Canonical Reliability Polynomial. Each bit in the AUXVEC represents a unique Partial Result which is generated during the initial phases of the program run and stored away in a hash table. Page 60.

NCER Neighbors Class Equivalence Relation; used to detect the symmetry classes of a graph. Page 70.

NCG Neighbors Class Graph; its vertices correspond to the equivalence classes generated by the NCER equivalence relation on the PMS graph G. Page 73.

EDS "Equal degree then split" partition; formed by the NCER equivalence relation on the PMS graph G. Page 74.

TNCER Typed Neighbors Class Equivalence Relation; a modified version of the Neighbors Class Equivalence Relation (NCER). Page 79.

ETEDS "Equal Type then Equal Degree then Split", this is the name of an algorithm to discover symmetry classes of the PMS graph based on the Typed Neighbors Class Equivalence Relation (TNCER) Page 80.

Pendant Tree Subgraph (PTS)
A Pendant Tree Subgraph is a maximal rooted tree subgraph of G such that the root vertex of the tree is an articulation vertex of G and the simple path, p_{xy}, between any pair of vertices v_x and v_y in the tree is unique in G. Page 91.

Kernel The Kernel is the subgraph of G which remains when the Pendant Tree Subgraphs of G have been removed. Page 115.

Partial Results
Partial Results are Canonical Reliability Polynomials which are generated due to the application of fragments of the original input requirements to the various Segments of the PMS structure Page 117.

Segments Table
This is used to retain information about the various segments into which the PMS graph G was divided. Page 122.

Critical Components
Components in the PMS structure whose component type appears in the requirements expression. Page 124.

Auxiliary Components
All components in the PMS structure which are not Critical Components. Page 124.

m-composition of the integer N
Page 125.

Capacity Vector
This conveys the number of components of some given type present in the various segments of G. Page 125.

Feasible Compositions
Page 127.

Infeasible Compositions
Page 127.

Fragment Requirement
Page 128.

Canonical Reliability Polynomial Tree (CRPTree)
The CRPTree is a tree constructed by ADVISER during the running of the OVERLORD routine. Its vertices are labelled with Partial Result CRPs and "collapsing" it gives the System CRP. Page 134.

Templates Table
When symmetric substructures exist in the PMS structure, the application of a given requirement to a symmetric group results in Partial Result CRPs which are also similar. Such similar CRPs may be represented by a single template. The Templates Table holds all such templates which were generated by ADVISER during a run.
Page 137.

Communicability Graph
Page 141.

Communicability Edge
See Communicability Graph. Page 141.

Compositions Table
The Compositions Table is very important in the operation of ADVISER and is used to cycle through all possible compositions of the given minimal requirements in order to determine the cases in which system success occurs. Page 150.

Requirements Array
This array holds those atoms of given minimal requirements which are currently being processed together as a Conjunctive Requirement by the OVERLORD routine.
Page 150.

Currently Chosen Kernel Set (CCKS)
Page 155.

Row-CRP
Page 159.

Kernel-CRP
Page 159.

Internal Port Connection Matrix (IPCM)
The IPCM for a particular component in the structure describes for the purposes of the Side Constraints which ports of the component are able to communicate information through the internals of the component Page 166.

Bounded Clustering of Critical Components
Page 171.

References and Bibliography

[Aggarwal 75a] Aggarwal, K.K., et al.
Computational time and absolute error comparisons for reliability expressions derived by various methods.
Microelectronics and Reliability 14:465-467, 1975.

[Aggarwal 75b] Aggarwal, K.K.
A fast algorithm for reliability evaluation.
IEEE Transactions on Reliability R-24:83-85, April, 1975.

[Aggarwal 78] Aggarwal, K.K., and Rai, S.
Symbolic Reliability Evaluation Using Logical Signal Relations.
IEEE Transactions on Reliability R-27(3):202-205, August, 1978.

[Avizienis 75] Avizienis, A.
Architecture of Fault-Tolerant Computing Systems.
In *Proceedings of the Fifth Annual International Symposium on Fault-Tolerant Computing*, pages 3-16. IEEE Computer Society, 1975.

[Ball 80] Ball, M.O.
Complexity of network reliability computations.
Networks 10:153-165, 1980.

[Barlow 75a] Barlow, R.E., and Proschan, F.
Statistical Theory of Reliability and Life Testing.
Holt, Rinehart and Winston, 1975.

[Barlow 75b] Barlow, R.E., and Lambert, H.E.
Introduction to Fault Tree Analysis.
In Barlow, R.E. (editor), *Reliability and Fault Tree Analysis: Theoretical and Applied Aspects of System Reliability and Safety Assessment*, pages 7-35. Soc. Indust. and Appl. Math., Philadelphia, 1975.

[Barlow 76] Barlow, R.E., and Proschan, F.
Some current academic research in system reliability theory.
IEEE Transactions on Reliability R-25(3):198, 1976.

[Beaudry 78] Beaudry, M.D.
Performance-related reliability measures for Computing Systems.
IEEE Transactions on Computers C-27(6):540-547, June, 1978.

[Bell 71] Bell, C.G., and Newell, A.
Computer Structures: Readings and Examples.
McGraw-Hill, 1971.

[Bennetts 75] Bennetts, R.G.
On the analysis of fault trees.
IEEE Transactions on Reliability R-24:175, 1975.

[Boesch 72] Boesch, F.T., and Felzer, A.P.
A general class of invulnerable graphs.
Networks 2:261-283, 1972.

[Bouricius 69] Bouricius, W.G, Carter, W.C. and Schneider, P.R.
Reliability Modeling Techniques for Self-Repairing Computer Systems.
In *Proc. 24th. National Conference ACM*, pages 295-309. Association for Computing Machinery, 1969.

[Bouricius 71] Bouricius, W.G., Carter, W.C., Jessep, D.C., Schneider, P.R. and Wadia A.B.
Reliability Modeling for Fault-Tolerant Computers.
IEEE Transactions on Computers C-20(11):1306-1311, November, 1971.

[Brown 71] Brown, D.B.
A computerized algorithm for determining the reliability of redundant configurations.
IEEE Transactions on Reliability R-20(3):108, August, 1971.

[Buzacott 67] Buzacott, J.
Finding the MTBF of repairable systems by reduction of the reliability block diagram.
Microelectronics and Reliability 6:105-112, 1967.

[Buzacott 70] Buzacott, J.
Network approaches to finding the reliability of repairable systems.
IEEE Transactions on Reliability R-19(4):140, November, 1970.

[Camarda 78] Camarda, P., Corsi, F., and Trentadue, A.
An efficient simple algorithm for Fault Tree automatic synthesis.
IEEE Transactions on Reliability R-27(3):215-221, August, 1978.

[Castillo 80] Castillo, X. and Siewiorek, D.P.
A performance-reliability model for computing systems.
In *Digest of Papers, FTCS-10: Tenth International Symposium on Fault-Tolerant Computing*, pages 187-192. IEEE Computer Society, October, 1980.

[Chatterjee 75] Chatterjee, P.
Modularization of Fault Trees: A method to reduce the cost of analysis.
In Barlow, R.E. (editor), *Reliability and Fault Tree Analysis: Theoretical and Applied Aspects of System Reliability and Safety Assessment*, pages 37-56. Soc. Indust. and Appl. Math., Philadelphia, 1975.

[Chelson 71] Chelson, P.O. and Eckstein, R.E.
Reliability Computation from Reliability Block Diagrams.
Technical Report 32-1543, National Aeronautics And Space Administration, Jet Propulsion Laboratory, Pasadena, Ca., December, 1971.

[Chung 71] Chung, W.K.
Generalized reliability function for systems of arbitrary complexity.
IEEE Transactions on Reliability R-20(2):85, 1971.

[Cox 68] Cox, R.E., and Miller, H.D.
The theory of stochastic processes.
Methuen and Co., London, 1968.

[Creasey 67] Creasey, D.J.
Reliability predictions for repairable systems containing redundancy.
Microelectronics and Reliability 6:135-142, 1967.

[Fleming 71] Fleming, J.L.
RELCOMP: A computer program for calculating system reliability and MTBF.
IEEE Transactions on Reliability R-20(3):102, August, 1971.

[Ford 62] Ford, L.K. and Fulkerson, D.R.
Flows in Networks.
Princeton University Press, Princeton, N.J., 1962.

[Frank 70] Frank, H., and Frisch, I.T.
Analysis and design of survivable networks.
IEEE Transactions on Communications Technology COM-18(5):501-519, 1970.

[Fratta 75] Fratta, L. and Montanari, U.
A Vertex Elimination Algorithm for Enumerating all Simple Paths in a Graph.
Networks 5:151-177, 1975.

[Fussell 74] Fussell, J.B., Powers, G.J. and Bennetts, R.G.
Fault Trees -- A state of the art discussion.
IEEE Transactions on Reliability R-23:51, 1974.

[Fussell 75a] Fussell, J.B.
Computer Aided Fault Tree Construction for Electrical Systems.
In Barlow, R.E. (editor), *Reliability and Fault Tree Analysis: Theoretical and Applied Aspects of System Reliability and Safety Assessment*, pages 37-56. Soc. Indust. and Appl. Math., Philadelphia, 1975.

[Fussell 75b] Fussell, J.B.
How to hand calculate system reliability and safety characteristics.
IEEE Transactions on Reliability R-24:169, 1975.

[Gandhi 72] Gandhi, S.L., Inoue, K., and Henley, E.J.
Computer aided system reliability analysis and optimization.
In Vlietstra, J. and Wielinga, R.F. (editors), *Computer-Aided Design: Proc IFIP Working Conference on Principles of Computer-Aided Design*, pages 283-308. IFIP, Eindhoven, Oct, 1972.

[Gaschnig 77] Gaschnig, J.
A "Neighbors Class" Node Partitioning Algorithm for Finding Symmetry Classes in Graphs.
1977.
Draft, September 18, Unpublished.

[Greene 68] Greene, K. and Cunningham, T.J.
Failure modes, effects and criticality analysis.
In *Proceedings, 1968 Annual Symposium on Reliability*, pages 374. IEEE, Boston, 1968.

[Hansler 74] Hansler, E., McAuliffe, G.K. and Wilkov, R.S.
Exact Calculation of Computer Network Reliability.
Networks 4:95-112, 1974.

[Hill 68] Hill, F.J. and Peterson, G.R.
Introduction to Switching Theory and Logical Design.
John Wiley & Sons, New York, 1968.

[Hopcroft 73] Hopcroft, J. and Tarjan, R.
Algorithm 447: Efficient Algorithms for Graph Manipulation.
Communications Of The ACM 16(6):372-378, June, 1973.

[Jenny 69] Jenny, J.A.
The effect of partial failure modes on reliability analysis.
IEEE Transactions on Reliability R-18(4):175, November, 1969.

[Katzman 77] Katzman, J.A.
A Fault-Tolerant Computing System.
Technical Report, Tandem Computers Inc., 1977.

[Kim 72] Kim, Y.H., Case, K.E., and Ghare, P.M.
A method for computing complex system reliability.
IEEE Transactions on Reliability R-21(2):215, May, 1972.

[Knight 75] Knight, L.
The measurement and prediction of the reliability of computing systems.
In *Proc. Internepcon (Microelectron.)*, pages 205. IEEE (?), October, 1975.

[Knudsen 73] Knudsen, M.J.
PMSL, An Interactive Language for System-Level Description and Analysis of Computer Structures.
PhD thesis, Carnegie-Mellon University, April, 1973.

[Knuth 69] Knuth, D.E.
The Art of Computer Programming. Volume 2: *Seminumerical Algorithms.*
Addison Wesley, 1969.

[Knuth 75a] Knuth, D.E.
The Art of Computer Programming. Volume 3: *Sorting and Searching*.
Addison Wesley, 1975.

[Knuth 75b] Knuth, D.E.
The Art of Computer Programming. Volume 1: *Fundamental Algorithms*.
Addison Wesley, 1975.
Second Edition.

[Koen 74] Koen, B.V. and Carnino, A.
Reliability Calculation with a List Processing Technique.
IEEE Transactions on Reliability R-23:43, 1974.

[Krishnamurthy 72]
Krishnamurthy, E.V. and Komissar, G.
Computer-Aided Reliability Network Analysis.
IEEE Transactions on Reliability R-21(2):86, May, 1972.

[Landrault 78] Landrault, C. and Laprie, J.C.
SURF -- A Program for Modeling and Reliability Prediction for Fault-Tolerant Computing Systems.
In J. Moneta (editor), *Information Technology*, . North-Holland Publishing Co., 1978.

[Lapp 77] Lapp, S., and Powers, G.
Computer-Aided Synthesis of Fault-Trees.
IEEE Transactions on Reliability R-26(1):2, April, 1977.

[Laprie 76] Laprie, J.C.
On the reliability prediction of repairable redundant digital systems.
IEEE Transactions on Reliability R-25(4):256, October, 1976.

[Lin 69] Lin, P.M. and Alderson, G.E.
Symbolic Network Functions by a Single Path-Finding Algorithm.
In *Proc. of 7th Allerton Conference on Circuit and System Theory*, pages 196. IEEE (?), 1969.

[Lin 76] Lin, P.M., Leon, B.J., and Huang, T.C.
A new algorithm for symbolic reliability analysis.
IEEE Transactions on Reliability R-25(1):2-15, April, 1976.

[Liu 68] Liu, C.L.
Introduction to Combinatorial Mathematics.
McGraw Hill, 1968.

[Locks 71] Locks, M.O.
The maximum error in system reliability calculation by using a subset of the minimal states.
IEEE Transactions on Reliability R-20(4):231, November, 1971.

[Locks 80] Locks, M.O.
Recursive Disjoint Products, Inclusion-Exclusion and Min-Cut Approximations.
IEEE Transactions on Reliability R-29(5):368-371, December, 1980.

[Macsyma 77] The MathLab Group.
MACSYMA Reference Manual
Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Mass., 1977.
Version 9, Second Printing, December 1977.

[Mathur 72] Mathur, F.P.
Automation of reliability evaluation procedures through CARE -- The computer-aided reliability estimation program.
In *Proceedings Fall Joint Computer Conference*, pages 65-77. AFIPS, 1972.

[Mathur 75a] Mathur, F.P. and deSouza, P.T.
Reliability modelling and analysis of general modular redundant systems.
IEEE Transactions on Reliability R-24(5):296, December, 1975.

[Mathur 75b] Mathur, F.P. and deSouza, P.T.
Reliability models of NMR systems.
IEEE Transactions on Reliability R-24:108, 1975.

[MIL-HDBK-217B 74]
Military Standardization Handbook: Reliability Prediction of Electronic Equipment
, September 1974.

[Misra 70a] Misra, K.B.
An Algorithm for the Reliability Evaluation of Redundant Networks.
IEEE Transactions on Reliability R-19:146-151, 1970.

[Misra 70b] Misra, K.B., Rao, T.M.S.
Reliability Analysis of Redundant Networks using Flowgraphs.
IEEE Transactions on Reliability R-19:19, February, 1970.

[Nelson 70] Nelson Jr., A.C., Batts, J.R., Beadles, R.L.
A computer program for approximating system reliability.
IEEE Transactions on Reliability R-19:61-65, May, 1970.

[Ng 77] Ng, Y.W., and Avizienis, A.
ARIES -- An Automated Reliability Estimation System for Redundant Digital Structures.
In *Proceedings 1977 Annual Reliability and Maintainability Symposium*, pages 108-113. IEEE, January, 1977.

[Ng 80] Ng, Y.W. and Avizienis, A.
A Unified Reliability Model for Fault-Tolerant Computers.
IEEE Transactions on Computers C-29(11):1002-1011, November, 1980.

[Nijenhuis 78] Nijenhuis, A. and Wilf, H.S.
Combinatorial Algorithms for Computers and Calculators, 2nd Edition.
Academic Press, New York, 1978.

[Ornstein 75] Ornstein, S.M., Crowther, W.R., et al.
Pluribus -- a reliable multiprocessor.
In *AFIPS Conference Proceedings*, pages 551-559. AFIPS, 1975.

[Osaki 76] Osaki, S. and Nakagawa, T.
Bibliography for reliability and availability for stochastic systems.
IEEE Transactions on Reliability R-25(4):284, October, 1976.

[Powers 76] Powers, G., and Lapp, S.
Computer-Aided Fault-Tree Synthesis.
Chemical Engineering Progress , April, 1976.

[Reiser 76] Reiser, J.F. (ed.).
SAIL
Stanford University, Stanford, California, 1976.
Computer Science Department Report No. STAN-CS-76-574, August 1976.
Also available from the National Technical Information Service, Springfield, Virginia, 22161.

[Rosenthal 77] Rosenthal, A.
Computing the reliability of complex networks.
SIAM J. Appl. Math. 32(2):384-393, March, 1977.

[Satyanarayana 78]
Satyanarayana, A., and Prabhakar, A.
New Topological Formula and Rapid Algorithm for Reliability Analysis of Complex Networks.
IEEE Transactions on Reliability R-27(2):82-100, June, 1978.

[Schick 78] Schick, G.J., and Wolverton, R.W.
An analysis of competing software reliability models.
IEEE Transactions on Software Engineering SE-4(2):104-120, March, 1978.

[Schroeder 70] Schroeder, R.J.
Fault Trees for Reliability Analysis.
In *Proceedings, 1970 Annual Symposium on Reliability*, pages 198. IEEE, February, 1970.
Los Angeles, 1970, IEEE Cat. # 70C2-R.

[Sharma 76] Sharma, J.
Algorithm for reliability evaluation of a reducible network.
IEEE Transactions on Reliability R-25(5):337, December, 1976.

[Shooman 68] Shooman, M.L.
Probabilistic Reliability: An Engineering Approach.
McGraw-Hill, New York, 1968.

[Shooman 70] Shooman, M.L.
The equivalence of reliability diagrams and fault-tree analysis.
IEEE Transactions on Reliability R-19(2):74-75, May, 1970.

[Siewiorek 78] Siewiorek, D.P. and Thomas, D.E. (eds.).
The Analysis of the Performance, Reliability and Life Cycle Cost of Multi-Processor Architectures and their Impact on SENET.
Research Report CMU-CS-78-126, Carnegie-Mellon University, Pittsburgh, PA, May, 1978.

[Staley 74] Staley, J.E. and Sutcliffe, P.S.
Reliability Block Diagram Analysis.
Microelectronics and Reliability 13(1):33-47, 1974.

[Stoffel 68] Stoffel, R.W.
System Analysis via Probability Diagrams.
In *Proceedings, 7th. Annual Reliability and Maintainability Conference*, pages 234. 1968.
San Francisco, CA, 1968.

[Swan 77] Swan, R.J., Fuller, S.H. and Siewiorek, D.P.
Cm*: A modular, multi-microprocessor.
In *AFIPS Conference Proceedings*, pages 637-644. AFIPS, 1977.
Volume 46.

[Tarjan 72] Tarjan, R.
Depth-First Search and Linear Graph Algorithms.
SIAM Journal of Computing 1(2):146-160, 1972.

[Teitelman 78] Teitelman, W. et al.
INTERLISP Reference Manual
Xerox Palo Alto Research Center, Palo Alto, CA., 1978.

[Tung 76] Tung, S.S.
Reliability of a tree network.
IEEE Transactions on Reliability R-25(5):333, December, 1976.

[USNRC 75] U.S.N.R.C.
Reactor Safety Study -- An Assessment of Accident Risks in U.S. Commercial Nuclear Power Plants, WASH1400 (NUREG-75/014).
Technical Report, U.S. Nuclear Regulatory Commission, Washington, D.C., 1975.
Available from NTIS, Springfield, VA, 22161.

[Widawsky 71] Widawsky, W.H.
Reliability and maintainability parameters evaluated with simulation.
IEEE Transactions on Reliability R-20(3):158, August, 1971.

[Wiesen 67] Wiesen, J.M.
Statistical methods in Reliability Analysis.
Electro-Technology :57, May, 1967.

[Wilkov 72] Wilkov, R.
Analysis and design of reliable computer networks.
IEEE Transactions on Communication COM-20(3):660, 1972.

[Worrell 76] Worrel, R.B. and Burdick, G.R.
Qualitative analysis in reliability and safety studies.
IEEE Transactions on Reliability R-25(3):164, August, 1976.

[Wulf 71] Wulf, W.A., et al.
BLISS Reference Manual: A Basic Language for Implementation of System Software for the PDP-10
Carnegie-Mellon University, 1971.

[Wulf 72] Wulf, W.A. and Bell, C.G.
C.mmp -- a multi-mini-processor.
In *Proc. AFIPS Fall Joint Computing Conference*, pages 765-777. AFIPS Press, Montvale, N.J., 1972.
Volume 41.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER CMU-CS-81-121	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) AUTOMATIC GENERATION OF RELIABILITY FUNCTIONS FOR PROCESSOR-MEMORY-SWITCH STRUCTURES		5. TYPE OF REPORT & PERIOD COVERED Interim 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Vittal Kini		8. CONTRACT OR GRANT NUMBER(s) N00014-77-C-0103
9. PERFORMING ORGANIZATION NAME AND ADDRESS Carnegie-Mellon University Computer Science Department Pittsburgh, PA. 15213		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS		12. REPORT DATE February 1981 13. NUMBER OF PAGES 286
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/ DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report)		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) Approved for public release; distribution unlimited		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number)		

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)


DATE
ILME